

---

**Novus**

**Kae Bartlett**

**Apr 25, 2024**



**CONTENTS:**

<b>1</b>	<b>API Reference</b>	<b>3</b>
1.1	Models . . . . .	3
1.2	ABCs . . . . .	150
1.3	Enums . . . . .	150
1.4	Flags . . . . .	159
1.5	Utils . . . . .	164
1.6	API . . . . .	165
<b>2</b>	<b>Client</b>	<b>177</b>
2.1	Client Quickstart . . . . .	177
2.2	Command-Line Interface . . . . .	177
2.3	API Reference . . . . .	178
<b>3</b>	<b>Configuration</b>	<b>191</b>
	<b>Index</b>	<b>193</b>



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*.



## 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 associated 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 associated 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 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.IOBBase* / *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.IOBBase* / *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.IOBBase* / *None*) – The discovery splash for the guild. Can be its bytes, a file path, or a file object.
- **banner** (*str* / *bytes* / *io.IOBBase* / *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 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 separately 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 separately 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 sticker.
- **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.
- **Returns**
- -----
- **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 or 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 against.
- **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 you 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 want to 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 convertible 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 you 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.IOBBase*) – 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 separately in the sidebar.
- **icon** (*str* | *bytes* | *io.IOBBase* | *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 sticker.
- **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.
- **Returns**
- -----
- **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 separately 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 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 or 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 against.
- **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 you 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 separately 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 sticker.
- **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 def 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 separately 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 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 or 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 against.
- **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**

Whether 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 convertible 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`

**vanity\_url\_code**

The guild's vanity 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 you 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 separately 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 sticker.
- **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 def 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 separately 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 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 or 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 against.
- **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 separately 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 type 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 sticker.
- **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.
- **Returns**
- -----
- **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 activities 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 interact 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 easy 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]`) – The 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**

The 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 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.ContextCommandData*(\*, *parent*: *Interaction*, *data*: *payloads.ApplicationCommandData*)

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**

MENTIONABLE

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.Intent
```

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: HTTP-Connection)

**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 in the channel that are public.

**async list\_private\_archived\_threads**(channel\_id: *int*, \*, before: *datetime* = *MISSING*, limit: *int* =  
*MISSING*) → *list*[*Channel*]

Get the archived threads in the 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 in the 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.



## 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 that you 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*, \*\*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 command 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 invocations 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[ContextCommandData], *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 invocations 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 guaranteed 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.

## 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`



## Symbols

\_\_getitem\_\_() (novus.ActionRow method), 126  
 \_\_getitem\_\_() (novus.StringSelectMenu method), 129  
 \_\_iter\_\_() (novus.ActionRow method), 126  
 \_\_iter\_\_() (novus.StringSelectMenu method), 130  
 \_\_setitem\_\_() (novus.ActionRow method), 126  
 \_\_setitem\_\_() (novus.StringSelectMenu method), 129

## A

accent\_color (novus.GuildMember attribute), 55  
 accent\_color (novus.User attribute), 110  
 accepted (novus.TeamMember attribute), 108  
 ACTION\_ROW (novus.ComponentType attribute), 154  
 action\_type (novus.AuditLogEntry attribute), 8  
 ActionRow (class in novus), 125  
 actions (novus.AutoModerationRule attribute), 10  
 ACTIVE (novus.EventStatus attribute), 154  
 active\_developer (novus.UserFlags attribute), 162  
 activities (novus.User attribute), 111  
 Activity (class in novus), 3  
 activity (novus.Message attribute), 77  
 ActivityType (class in novus), 159  
 add() (novus.ActionRow method), 125  
 add() (novus.StringSelectMenu method), 128  
 add\_command() (novus.ext.client.Client method), 179  
 add\_description() (novus.ext.client.CommandGroup method), 186  
 add\_field() (novus.Embed method), 123  
 add\_guild\_member() (novus.api.guild.GuildHTTPConnection method), 171  
 add\_guild\_member\_role()  
   (novus.api.guild.GuildHTTPConnection method), 171  
 add\_id() (novus.ext.client.Command method), 184  
 add\_id() (novus.ext.client.CommandGroup method), 186  
 add\_member() (novus.BaseGuild method), 23  
 add\_member() (novus.Guild method), 41  
 add\_member() (novus.GuildPreview method), 62  
 add\_member() (novus.PartialGuild method), 84  
 add\_member\_role() (novus.BaseGuild method), 24  
 add\_member\_role() (novus.Guild method), 42  
 add\_member\_role() (novus.GuildPreview method), 62  
 add\_member\_role() (novus.PartialGuild method), 84  
 add\_plugin() (novus.ext.client.Client method), 180  
 add\_plugin\_file() (novus.ext.client.Client method), 181  
 add\_reaction() (novus.Message method), 80  
 add\_reaction() (novus.WebhookMessage method), 117  
 add\_reactions (novus.Permissions attribute), 161  
 add\_role() (novus.GuildMember method), 59  
 add\_thread\_member()  
   (novus.api.channel.ChannelHTTPConnection method), 170  
 add\_thread\_member() (novus.Channel method), 32  
 administrator (novus.Permissions attribute), 161  
 AFK (novus.Status attribute), 158  
 afk\_channel\_id (novus.Guild attribute), 37  
 after (novus.AuditLogEntry attribute), 8  
 AGE\_RESTRICTED (novus.NSFWLevel attribute), 157  
 all() (novus.AllowedMentions class method), 119  
 ALL\_MEMBERS (novus.ContentFilterLevel attribute), 154  
 ALL\_MESSAGES (novus.NotificationLevel attribute), 157  
 allow (novus.PermissionOverwrite attribute), 125  
 AllowedMentions (class in novus), 119  
 animated (novus.Asset attribute), 6  
 animated (novus.Emoji attribute), 33  
 animated (novus.PartialEmoji attribute), 82  
 ANNOUNCEMENT\_THREAD (novus.ChannelType attribute), 153  
 APIIterator (class in novus.api), 167  
 APNG (novus.StickerFormat attribute), 157  
 app\_permissions (novus.Interaction attribute), 142  
 Application (class in novus), 4  
 application (novus.Message attribute), 78  
 application\_command (novus.ext.client.Command attribute), 184  
 application\_command  
   (novus.ext.client.CommandGroup attribute), 185  
 APPLICATION\_COMMAND (novus.InteractionType attribute), 156  
 APPLICATION\_COMMAND\_AUTOCOMPLETE\_RESULT  
   (novus.InteractionResponseType attribute), 155

- `application_command_badge` (*novus.ApplicationFlags* attribute), 160
- `application_command_permission_update` (*novus.AuditLogEntryType* attribute), 164
- `APPLICATION_COMMAND_PERMISSION_UPDATE` (*novus.AuditLogEventType* attribute), 151
- `application_id` (*novus.ApplicationCommand* attribute), 137
- `application_id` (*novus.Guild* attribute), 38
- `application_id` (*novus.Interaction* attribute), 141
- `application_id` (*novus.Message* attribute), 78
- `application_role_connection_metadata` (*novus.api.HTTPConnection* attribute), 166
- `ApplicationCommand` (class in *novus*), 136
- `ApplicationCommandChoice` (class in *novus*), 138
- `ApplicationCommandData` (class in *novus*), 148
- `ApplicationCommandOption` (class in *novus*), 138
- `ApplicationCommandType` (class in *novus*), 150
- `ApplicationFlags` (class in *novus*), 160
- `ApplicationOptionType` (class in *novus*), 150
- `ApplicationRoleHTTPConnection` (class in *novus.api.application\_role\_connection\_metadata*), 167
- `approximate_member_count` (*novus.Guild* attribute), 40
- `approximate_member_count` (*novus.GuildPreview* attribute), 61
- `approximate_presence_count` (*novus.Guild* attribute), 40
- `approximate_presence_count` (*novus.GuildPreview* attribute), 62
- `Asset` (class in *novus*), 6
- `asset` (*novus.Emoji* attribute), 33
- `asset` (*novus.PartialEmoji* attribute), 82
- `asset` (*novus.Sticker* attribute), 105
- `attach_files` (*novus.Permissions* attribute), 161
- `Attachment` (class in *novus*), 6
- `ATTACHMENT` (*novus.ApplicationOptionType* attribute), 150
- `attachments` (*novus.InteractionResolved* attribute), 150
- `attachments` (*novus.Message* attribute), 77
- `audit_log` (*novus.api.HTTPConnection* attribute), 166
- `AuditLog` (class in *novus*), 7
- `AuditLogContainer` (class in *novus*), 7
- `AuditLogEntry` (class in *novus*), 8
- `AuditLogEntryType` (class in *novus*), 162
- `AuditLogEventType` (class in *novus*), 151
- `AuditLogHTTPConnection` (class in *novus.api.audit\_log*), 168
- `author` (*novus.Embed* attribute), 121
- `author` (*novus.Message* attribute), 76
- `AUTO` (*novus.VideoQualityMode* attribute), 155
- `auto_moderation` (*novus.api.HTTPConnection* attribute), 166
- `AUTO_MODERATION_ACTION` (*novus.MessageType* attribute), 156
- `auto_moderation_block_message` (*novus.AuditLogEntryType* attribute), 164
- `AUTO_MODERATION_BLOCK_MESSAGE` (*novus.AuditLogEventType* attribute), 151
- `auto_moderation_configuration` (*novus.Intent* attribute), 159
- `auto_moderation_execution` (*novus.Intent* attribute), 160
- `auto_moderation_flag_to_channel` (*novus.AuditLogEntryType* attribute), 164
- `AUTO_MODERATION_FLAG_TO_CHANNEL` (*novus.AuditLogEventType* attribute), 151
- `auto_moderation_rule_create` (*novus.AuditLogEntryType* attribute), 164
- `AUTO_MODERATION_RULE_CREATE` (*novus.AuditLogEventType* attribute), 151
- `auto_moderation_rule_delete` (*novus.AuditLogEntryType* attribute), 164
- `AUTO_MODERATION_RULE_DELETE` (*novus.AuditLogEventType* attribute), 151
- `auto_moderation_rule_update` (*novus.AuditLogEntryType* attribute), 164
- `AUTO_MODERATION_RULE_UPDATE` (*novus.AuditLogEventType* attribute), 151
- `auto_moderation_user_communication_disabled` (*novus.AuditLogEntryType* attribute), 164
- `AUTO_MODERATION_USER_COMMUNICATION_DISABLED` (*novus.AuditLogEventType* attribute), 151
- `autocomplete` (*novus.ApplicationCommandOption* attribute), 141
- `AUTOCOMPLETE` (*novus.InteractionType* attribute), 156
- `autocomplete()` (*novus.ext.client.Command* method), 185
- `autocomplete()` (*novus.ext.client.CommandGroup* method), 187
- `AutoModerationAction` (class in *novus*), 9
- `AutoModerationActionType` (class in *novus*), 152
- `AutoModerationEventType` (class in *novus*), 153
- `AutoModerationHTTPConnection` (class in *novus.api.auto\_moderation*), 168
- `AutoModerationKeywordPresetType` (class in *novus*), 153
- `AutoModerationRule` (class in *novus*), 9
- `AutoModerationTriggerMetadata` (class in *novus*), 13
- `AutoModerationTriggerType` (class in *novus*), 153
- `autostart` (*novus.ext.client.Loop* attribute), 188
- `available` (*novus.Emoji* attribute), 33
- `available` (*novus.Sticker* attribute), 105
- `avatar` (*novus.GuildMember* attribute), 55
- `avatar` (*novus.User* attribute), 109
- `avatar` (*novus.Webhook* attribute), 114
- `avatar_hash` (*novus.GuildMember* attribute), 55

avatar\_hash (novus.User attribute), 109  
 avatar\_hash (novus.Webhook attribute), 114

## B

ban() (novus.BaseGuild method), 25  
 ban() (novus.Guild method), 42  
 ban() (novus.GuildMember method), 60  
 ban() (novus.GuildPreview method), 63  
 ban() (novus.PartialGuild method), 85  
 ban\_members (novus.Permissions attribute), 161  
 banner (novus.Guild attribute), 39  
 banner (novus.GuildMember attribute), 55  
 banner (novus.PartialGuild attribute), 83  
 banner (novus.User attribute), 110  
 banner\_hash (novus.Guild attribute), 39  
 banner\_hash (novus.GuildMember attribute), 55  
 banner\_hash (novus.PartialGuild attribute), 83  
 banner\_hash (novus.User attribute), 110  
 BaseGuild (class in novus), 13  
 before (novus.AuditLogEntry attribute), 8  
 before() (novus.ext.client.Loop method), 189  
 BLOCK\_MESSAGE (novus.AutoModerationActionType attribute), 153  
 BLURPLE (novus.ButtonStyle attribute), 153  
 BOOLEAN (novus.ApplicationOptionType attribute), 150  
 bot (novus.GuildMember attribute), 55  
 bot (novus.User attribute), 109  
 bot\_add (novus.AuditLogEntryType attribute), 163  
 BOT\_ADD (novus.AuditLogEventType attribute), 151  
 bot\_http\_interactions (novus.UserFlags attribute), 162  
 bot\_public (novus.Application attribute), 4  
 bot\_require\_code\_grant (novus.Application attribute), 4  
 bug\_hunter\_level\_1 (novus.UserFlags attribute), 162  
 bug\_hunter\_level\_2 (novus.UserFlags attribute), 162  
 bulk\_delete\_messages()  
   (novus.api.channel.ChannelHTTPConnection method), 169  
 bulk\_delete\_messages() (novus.Channel method), 31  
 burst (novus.Reaction attribute), 97  
 Button (class in novus), 126  
 BUTTON (novus.ComponentType attribute), 154  
 ButtonStyle (class in novus), 153

## C

CALL (novus.MessageType attribute), 156  
 callback (novus.ext.client.Command attribute), 184  
 cancel() (novus.ext.client.Loop method), 189  
 CANCELLED (novus.EventStatus attribute), 154  
 certified\_moderator (novus.UserFlags attribute), 162  
 change\_nickname (novus.Permissions attribute), 161

change\_presence() (novus.ext.client.Client method), 179  
 Channel (class in novus), 26  
 channel (novus.api.HTTPConnection attribute), 166  
 CHANNEL (novus.ApplicationOptionType attribute), 150  
 channel (novus.Interaction attribute), 142  
 channel (novus.Invite attribute), 74  
 channel (novus.Message attribute), 76  
 channel (novus.ScheduledEvent attribute), 99  
 channel (novus.VoiceState attribute), 112  
 channel\_create (novus.AuditLogEntryType attribute), 162  
 CHANNEL\_CREATE (novus.AuditLogEventType attribute), 151  
 channel\_delete (novus.AuditLogEntryType attribute), 162  
 CHANNEL\_DELETE (novus.AuditLogEventType attribute), 151  
 CHANNEL\_FOLLOW\_ADD (novus.MessageType attribute), 156  
 CHANNEL\_ICON\_CHANGE (novus.MessageType attribute), 156  
 channel\_id (novus.AutoModerationAction attribute), 9  
 channel\_id (novus.StageInstance attribute), 103  
 channel\_id (novus.Webhook attribute), 113  
 CHANNEL\_MESSAGE\_WITH\_SOURCE  
   (novus.InteractionResponseType attribute), 155  
 CHANNEL\_NAME\_CHANGE (novus.MessageType attribute), 156  
 channel\_overwrite\_create  
   (novus.AuditLogEntryType attribute), 162  
 CHANNEL\_OVERWRITE\_CREATE  
   (novus.AuditLogEventType attribute), 151  
 channel\_overwrite\_delete  
   (novus.AuditLogEntryType attribute), 163  
 CHANNEL\_OVERWRITE\_DELETE  
   (novus.AuditLogEventType attribute), 151  
 channel\_overwrite\_update  
   (novus.AuditLogEntryType attribute), 162  
 CHANNEL\_OVERWRITE\_UPDATE  
   (novus.AuditLogEventType attribute), 151  
 CHANNEL\_PINNED\_MESSAGE (novus.MessageType attribute), 156  
 CHANNEL\_SELECT (novus.ComponentType attribute), 154  
 channel\_types (novus.ApplicationCommandOption attribute), 140  
 channel\_update (novus.AuditLogEntryType attribute), 162  
 CHANNEL\_UPDATE (novus.AuditLogEventType attribute), 151  
 ChannelHTTPConnection (class in novus.api.channel), 168  
 channels (novus.InteractionResolved attribute), 150

- ChannelSelectMenu (class in novus), 130
- ChannelType (class in novus), 153
- CHAT\_INPUT (novus.ApplicationCommandType attribute), 150
- CHAT\_INPUT\_COMMAND (novus.MessageType attribute), 156
- choices (novus.ApplicationCommandOption attribute), 140
- chunk\_members() (novus.BaseGuild method), 16
- chunk\_members() (novus.Guild method), 42
- chunk\_members() (novus.GuildPreview method), 63
- chunk\_members() (novus.PartialGuild method), 85
- clear() (novus.ActionRow method), 126
- clear() (novus.StringSelectMenu method), 129
- clear\_fields() (novus.Embed method), 123
- clear\_reactions() (novus.Message method), 80
- clear\_reactions() (novus.WebhookMessage method), 117
- Client (class in novus.ext.client), 178
- close() (novus.ext.client.Client method), 181
- code (novus.Invite attribute), 74
- color (novus.Embed attribute), 120
- color (novus.Role attribute), 97
- Command (class in novus.ext.client), 183
- command() (in module novus.ext.client), 182
- command\_ids (novus.ext.client.Command attribute), 184
- command\_ids (novus.ext.client.CommandGroup attribute), 186
- CommandDescription (class in novus.ext.client), 185
- CommandError (class in novus.ext.client), 190
- CommandGroup (class in novus.ext.client), 185
- commands (novus.ext.client.Client attribute), 178
- commands (novus.ext.client.CommandGroup attribute), 186
- COMPETING (novus.ActivityType attribute), 159
- COMPLETED (novus.EventStatus attribute), 155
- Component (class in novus), 150
- component (novus.MessageComponentData attribute), 149
- components (novus.ActionRow attribute), 125
- components (novus.Message attribute), 78
- components (novus.ModalSubmitData attribute), 149
- ComponentType (class in novus), 154
- Config (class in novus.ext.client), 181
- config (novus.ext.client.Client attribute), 178
- connect (novus.Permissions attribute), 161
- connect() (novus.ext.client.Client method), 181
- connect\_webserver() (novus.ext.client.Client method), 181
- content (novus.Message attribute), 76
- content\_type (novus.File attribute), 124
- ContentFilterLevel (class in novus), 154
- CONTEXT\_MENU\_COMMAND (novus.MessageType attribute), 156
- ContextComandData (class in novus), 147
- cover\_image (novus.Application attribute), 5
- cover\_image\_hash (novus.Application attribute), 5
- create() (novus.AutoModerationRule class method), 12
- create() (novus.BaseGuild class method), 13
- create() (novus.Emoji class method), 34
- create() (novus.Guild class method), 43
- create() (novus.GuildPreview class method), 63
- create() (novus.Message class method), 79
- create() (novus.PartialGuild class method), 85
- create() (novus.ScheduledEvent class method), 100
- create() (novus.StageInstance class method), 103
- create() (novus.Sticker class method), 105
- create() (novus.WebhookMessage class method), 117
- create\_auto\_moderation\_rule() (novus.api.auto\_moderation.AutoModerationHTTPConnection method), 168
- create\_auto\_moderation\_rule() (novus.BaseGuild method), 16
- create\_auto\_moderation\_rule() (novus.Guild method), 43
- create\_auto\_moderation\_rule() (novus.GuildPreview method), 63
- create\_auto\_moderation\_rule() (novus.PartialGuild method), 85
- create\_channel() (novus.BaseGuild method), 17
- create\_channel() (novus.Guild method), 44
- create\_channel() (novus.GuildPreview method), 64
- create\_channel() (novus.PartialGuild method), 86
- create\_channel\_invite() (novus.api.channel.ChannelHTTPConnection method), 169
- create\_dm() (novus.api.user.UserHTTPConnection method), 174
- create\_dm\_channel() (novus.User method), 112
- create\_emoji() (novus.BaseGuild method), 19
- create\_emoji() (novus.Guild method), 45
- create\_emoji() (novus.GuildPreview method), 65
- create\_emoji() (novus.PartialGuild method), 87
- create\_guild() (novus.api.guild.GuildHTTPConnection method), 171
- create\_guild\_ban() (novus.api.guild.GuildHTTPConnection method), 172
- create\_guild\_channel() (novus.api.guild.GuildHTTPConnection method), 171
- create\_guild\_emoji() (novus.api.emoji.EmojiHTTPConnection method), 170
- create\_guild\_role() (novus.api.guild.GuildHTTPConnection method), 172
- create\_guild\_scheduled\_event() (novus.api.guild\_scheduled\_event.GuildEventHTTPConnection method), 172



method), 172  
 create\_guild\_sticker()  
     (novus.api.sticker.StickerHTTPConnection  
     method), 173  
 create\_instant\_invite (novus.Permissions at-  
     tribute), 160  
 create\_invite() (novus.Channel method), 29  
 create\_message() (novus.api.channel.ChannelHTTPConnection  
     method), 168  
 create\_overwrite() (novus.Channel method), 29  
 create\_private\_threads (novus.Permissions at-  
     tribute), 162  
 create\_public\_threads (novus.Permissions at-  
     tribute), 162  
 create\_reaction() (novus.api.channel.ChannelHTTPConnection  
     method), 168  
 create\_role() (novus.BaseGuild method), 19  
 create\_role() (novus.Guild method), 45  
 create\_role() (novus.GuildPreview method), 65  
 create\_role() (novus.PartialGuild method), 87  
 create\_scheduled\_event() (novus.BaseGuild  
     method), 20  
 create\_scheduled\_event() (novus.Guild method), 45  
 create\_scheduled\_event() (novus.GuildPreview  
     method), 66  
 create\_scheduled\_event() (novus.PartialGuild  
     method), 88  
 create\_stage\_instance()  
     (novus.api.stage\_instance.StageHTTPConnection  
     method), 173  
 create\_sticker() (novus.BaseGuild method), 22  
 create\_sticker() (novus.Guild method), 46  
 create\_sticker() (novus.GuildPreview method), 66  
 create\_sticker() (novus.PartialGuild method), 88  
 create\_thread() (novus.Channel method), 31  
 create\_thread() (novus.Message method), 81  
 create\_thread() (novus.WebhookMessage method),  
     118  
 create\_thread\_in\_forum() (novus.Channel method),  
     31  
 create\_webhook() (novus.api.webhook.WebhookHTTPConnection  
     method), 174  
 created\_at (novus.Invite attribute), 75  
 CREATION\_DATE (novus.ForumSortOrder attribute), 155  
 creator (novus.ScheduledEvent attribute), 99  
 creator\_id (novus.AutoModerationRule attribute), 10  
 crosspost() (novus.Message method), 80  
 crosspost() (novus.WebhookMessage method), 118  
 crosspost\_message()  
     (novus.api.channel.ChannelHTTPConnection  
     method), 168  
 CTA (novus.ButtonStyle attribute), 153  
 CUSTOM (novus.ActivityType attribute), 159  
 custom\_id (novus.Button attribute), 127  
 custom\_id (novus.ChannelSelectMenu attribute), 130  
 custom\_id (novus.MentionableSelectMenu attribute),  
     132  
 custom\_id (novus.MessageComponentData attribute),  
     149  
 custom\_id (novus.ModalSubmitData attribute), 149  
 custom\_id (novus.RoleSelectMenu attribute), 131  
 custom\_id (novus.StringSelectMenu attribute), 128  
 custom\_id (novus.TextInput attribute), 134  
 custom\_id (novus.UserSelectMenu attribute), 131  
 custom\_install\_url (novus.Application attribute), 6

## D

DANGER (novus.ButtonStyle attribute), 153  
 data (novus.File attribute), 124  
 data (novus.Interaction attribute), 141  
 deaf (novus.GuildMember attribute), 57  
 deaf (novus.VoiceState attribute), 113  
 deafen\_members (novus.Permissions attribute), 161  
 DEFAULT (novus.MessageType attribute), 156  
 DEFAULT (novus.NSFWLevel attribute), 157  
 default (novus.SelectOption attribute), 133  
 default\_member\_permissions  
     (novus.ApplicationCommand attribute), 137  
 default\_member\_permissions  
     (novus.PartialApplicationCommand attribute),  
     136  
 default\_message\_notifications (novus.Guild at-  
     tribute), 37  
 defer() (novus.GuildInteraction method), 145  
 defer() (novus.Interaction method), 143  
 defer\_update() (novus.GuildInteraction method), 145  
 defer\_update() (novus.Interaction method), 144  
 DEFERRED\_CHANNEL\_MESSAGE\_WITH\_SOURCE  
     (novus.InteractionResponseType attribute),  
     155  
 DEFERRED\_UPDATE\_MESSAGE  
     (novus.InteractionResponseType attribute),  
     155  
 delete() (novus.AutoModerationRule method), 12  
 delete() (novus.BaseGuild method), 15  
 delete() (novus.Channel method), 28  
 delete() (novus.Emoji method), 35  
 delete() (novus.Guild method), 46  
 delete() (novus.GuildPreview method), 67  
 delete() (novus.Invite method), 75  
 delete() (novus.Message method), 80  
 delete() (novus.PartialGuild method), 89  
 delete() (novus.Role method), 98  
 delete() (novus.ScheduledEvent method), 102  
 delete() (novus.Sticker method), 107  
 delete() (novus.WebhookMessage method), 116  
 delete\_all\_reactions()  
     (novus.api.channel.ChannelHTTPConnection

- method), 169
- `delete_all_reactions_for_emoji()` (*novus.api.channel.ChannelHTTPConnection method*), 169
- `delete_auto_moderation_rule()` (*novus.api.auto\_moderation.AutoModerationHTTPConnection method*), 168
- `delete_channel()` (*novus.api.channel.ChannelHTTPConnection method*), 168
- `delete_channel_permission()` (*novus.api.channel.ChannelHTTPConnection method*), 169
- `delete_guild()` (*novus.api.guild.GuildHTTPConnection method*), 171
- `delete_guild_emoji()` (*novus.api.emoji.EmojiHTTPConnection method*), 171
- `delete_guild_role()` (*novus.api.guild.GuildHTTPConnection method*), 172
- `delete_guild_scheduled_event()` (*novus.api.guild\_scheduled\_event.GuildEventHTTPConnection method*), 172
- `delete_guild_sticker()` (*novus.api.sticker.StickerHTTPConnection method*), 173
- `delete_invite()` (*novus.api.invite.InviteHTTPConnection method*), 173
- `delete_message()` (*novus.api.channel.ChannelHTTPConnection method*), 169
- `delete_original()` (*novus.GuildInteraction method*), 145
- `delete_original()` (*novus.Interaction method*), 144
- `delete_own_reaction()` (*novus.api.channel.ChannelHTTPConnection method*), 169
- `delete_role()` (*novus.BaseGuild method*), 20
- `delete_role()` (*novus.Guild method*), 46
- `delete_role()` (*novus.GuildPreview method*), 67
- `delete_role()` (*novus.PartialGuild method*), 89
- `delete_stage_instance()` (*novus.api.stage\_instance.StageHTTPConnection method*), 173
- `delete_user_reaction()` (*novus.api.channel.ChannelHTTPConnection method*), 169
- `delete_webhook()` (*novus.api.webhook.WebhookHTTPConnection method*), 174
- `delete_webhook_message()` (*novus.api.webhook.WebhookHTTPConnection method*), 175
- `deny` (*novus.PermissionOverwrite attribute*), 125
- `description` (*novus.Application attribute*), 4
- `description` (*novus.ApplicationCommand attribute*), 137
- `description` (*novus.ApplicationCommandOption attribute*), 139
- `description` (*novus.Embed attribute*), 120
- `description` (*novus.File attribute*), 124
- `description` (*novus.Guild attribute*), 39
- `description` (*novus.GuildPreview attribute*), 62
- `description` (*novus.PartialApplicationCommand attribute*), 135
- `description` (*novus.PartialGuild attribute*), 83
- `description` (*novus.ScheduledEvent attribute*), 99
- `description` (*novus.SelectOption attribute*), 133
- `description` (*novus.Sticker attribute*), 105
- `description_localizations` (*novus.ApplicationCommand attribute*), 137
- `description_localizations` (*novus.ApplicationCommandOption attribute*), 140
- `description_localizations` (*novus.PartialApplicationCommand attribute*), 136
- `direct_message_reactions` (*novus.Intents attribute*), 159
- `direct_message_ttyping` (*novus.Intents attribute*), 159
- `direct_messages` (*novus.Intents attribute*), 159
- `disabled` (*novus.Button attribute*), 127
- `disabled` (*novus.ChannelSelectMenu attribute*), 130
- `DISABLED` (*novus.ContentFilterLevel attribute*), 154
- `disabled` (*novus.MentionableSelectMenu attribute*), 133
- `disabled` (*novus.RoleSelectMenu attribute*), 131
- `disabled` (*novus.StringSelectMenu attribute*), 128
- `disabled` (*novus.UserSelectMenu attribute*), 132
- `discovery_splash` (*novus.Guild attribute*), 36
- `discovery_splash` (*novus.GuildPreview attribute*), 61
- `discovery_splash_hash` (*novus.Guild attribute*), 36
- `discovery_splash_hash` (*novus.GuildPreview attribute*), 61
- `discriminator` (*novus.GuildMember attribute*), 55
- `discriminator` (*novus.User attribute*), 109
- `DISPATCH` (*novus.GatewayOpcode attribute*), 155
- `dispatch()` (*novus.ext.client.Client method*), 181
- `dispatch()` (*novus.ext.client.Plugin method*), 182
- `DM` (*novus.ChannelType attribute*), 153
- `dm_permission` (*novus.ApplicationCommand attribute*), 138
- `dm_permission` (*novus.PartialApplicationCommand attribute*), 136
- `DND` (*novus.Status attribute*), 158
- `DO_NOT_DISTURB` (*novus.Status attribute*), 158
- `duration` (*novus.AutoModerationAction attribute*), 9
- ## E
- `edit()` (*novus.AutoModerationRule method*), 11
- `edit()` (*novus.BaseGuild method*), 14

- edit() (*novus.Channel* method), 27  
 edit() (*novus.Emoji* method), 35  
 edit() (*novus.Guild* method), 47  
 edit() (*novus.GuildMember* method), 59  
 edit() (*novus.GuildPreview* method), 67  
 edit() (*novus.Message* method), 80  
 edit() (*novus.PartialGuild* method), 89  
 edit() (*novus.Role* method), 98  
 edit() (*novus.ScheduledEvent* method), 102  
 edit() (*novus.StageInstance* method), 104  
 edit() (*novus.Sticker* method), 107  
 edit() (*novus.Webhook* method), 115  
 edit() (*novus.WebhookMessage* method), 117  
 edit\_channel\_permissions()  
     (*novus.api.channel.ChannelHTTPConnection*  
     method), 169  
 edit\_member() (*novus.BaseGuild* method), 24  
 edit\_member() (*novus.Guild* method), 48  
 edit\_member() (*novus.GuildPreview* method), 68  
 edit\_member() (*novus.PartialGuild* method), 90  
 edit\_message() (*novus.api.channel.ChannelHTTPConnection*  
     method), 169  
 edit\_original() (*novus.GuildInteraction* method),  
     145  
 edit\_original() (*novus.Interaction* method), 144  
 edit\_role() (*novus.BaseGuild* method), 20  
 edit\_role() (*novus.Guild* method), 49  
 edit\_role() (*novus.GuildPreview* method), 69  
 edit\_role() (*novus.PartialGuild* method), 91  
 edit\_webhook\_message()  
     (*novus.api.webhook.WebhookHTTPConnection*  
     method), 175  
 edit\_with\_token() (*novus.Webhook* method), 115  
 edited\_timestamp (*novus.Message* attribute), 76  
 ELEVATED (*novus.MFALevel* attribute), 156  
 email (*novus.GuildMember* attribute), 56  
 email (*novus.User* attribute), 110  
 Embed (class in *novus*), 119  
 embed\_links (*novus.Permissions* attribute), 161  
 embedded (*novus.ApplicationFlags* attribute), 160  
 embeds (*novus.Message* attribute), 77  
 Emoji (class in *novus*), 33  
 emoji (*novus.api.HTTPConnection* attribute), 166  
 emoji (*novus.Button* attribute), 127  
 emoji (*novus.ForumTag* attribute), 36  
 emoji (*novus.Reaction* attribute), 96  
 emoji (*novus.SelectOption* attribute), 133  
 emoji\_create (*novus.AuditLogEntryType* attribute),  
     163  
 EMOJI\_CREATE (*novus.AuditLogEventType* attribute),  
     151  
 emoji\_delete (*novus.AuditLogEntryType* attribute),  
     163  
 EMOJI\_DELETE (*novus.AuditLogEventType* attribute),  
     151  
 emoji\_update (*novus.AuditLogEntryType* attribute),  
     163  
 EMOJI\_UPDATE (*novus.AuditLogEventType* attribute),  
     151  
 EmojiHTTPConnection (class in *novus.api.emoji*), 170  
 emojis (*novus.Guild* attribute), 38  
 emojis (*novus.GuildPreview* attribute), 61  
 enabled (*novus.AutoModerationRule* attribute), 10  
 end (*novus.ext.client.LoopBehavior* attribute), 189  
 end\_behavior (*novus.ext.client.Loop* attribute), 188  
 end\_time (*novus.ScheduledEvent* attribute), 99  
 entity\_id (*novus.ScheduledEvent* attribute), 100  
 entity\_type (*novus.ScheduledEvent* attribute), 100  
 entries (*novus.AuditLog* attribute), 7  
 event() (in module *novus.ext.client*), 187  
 event\_id (*novus.StageInstance* attribute), 103  
 event\_type (*novus.AutoModerationRule* attribute), 10  
 EventEntityType (class in *novus*), 154  
 EventListener (class in *novus.ext.client*), 187  
 EventPrivacyLevel (class in *novus*), 154  
 EventStatus (class in *novus*), 154  
 execute\_webhook() (*novus.api.webhook.WebhookHTTPConnection*  
     method), 174  
 exempt\_channel\_ids (*novus.AutoModerationRule* at-  
     tribute), 10  
 exempt\_role\_ids (*novus.AutoModerationRule* at-  
     tribute), 10  
 EXPLICIT (*novus.NSFWLevel* attribute), 157  
 explicit\_content\_filter (*novus.Guild* attribute), 37  
 EXTERNAL (*novus.EventEntityType* attribute), 154
- ## F
- features (*novus.Guild* attribute), 38  
 features (*novus.GuildPreview* attribute), 61  
 features (*novus.PartialGuild* attribute), 83  
 fetch() (*novus.AuditLog* class method), 7  
 fetch() (*novus.AutoModerationRule* class method), 11  
 fetch() (*novus.BaseGuild* class method), 14  
 fetch() (*novus.Channel* class method), 27  
 fetch() (*novus.Emoji* class method), 34  
 fetch() (*novus.Guild* class method), 49  
 fetch() (*novus.GuildMember* class method), 58  
 fetch() (*novus.GuildPreview* class method), 69  
 fetch() (*novus.Invite* class method), 75  
 fetch() (*novus.Message* class method), 80  
 fetch() (*novus.PartialGuild* class method), 91  
 fetch() (*novus.ScheduledEvent* class method), 101  
 fetch() (*novus.StageInstance* class method), 104  
 fetch() (*novus.Sticker* class method), 106  
 fetch() (*novus.User* class method), 111  
 fetch() (*novus.Webhook* class method), 115  
 fetch() (*novus.WebhookMessage* class method), 116

- `fetch_active_threads()` (*novus.BaseGuild method*), 18
- `fetch_active_threads()` (*novus.Guild method*), 49
- `fetch_active_threads()` (*novus.GuildPreview method*), 70
- `fetch_active_threads()` (*novus.PartialGuild method*), 92
- `fetch_all_emojis()` (*novus.BaseGuild method*), 18
- `fetch_all_emojis()` (*novus.Guild method*), 49
- `fetch_all_emojis()` (*novus.GuildPreview method*), 70
- `fetch_all_emojis()` (*novus.PartialGuild method*), 92
- `fetch_all_for_guild()` (*novus.AutoModerationRule class method*), 11
- `fetch_all_for_guild()` (*novus.Emoji class method*), 35
- `fetch_all_for_guild()` (*novus.ScheduledEvent class method*), 101
- `fetch_all_for_guild()` (*novus.Sticker class method*), 106
- `fetch_all_stickers()` (*novus.BaseGuild method*), 21
- `fetch_all_stickers()` (*novus.Guild method*), 50
- `fetch_all_stickers()` (*novus.GuildPreview method*), 70
- `fetch_all_stickers()` (*novus.PartialGuild method*), 92
- `fetch_audit_logs()` (*novus.BaseGuild method*), 16
- `fetch_audit_logs()` (*novus.Guild method*), 50
- `fetch_audit_logs()` (*novus.GuildPreview method*), 70
- `fetch_audit_logs()` (*novus.PartialGuild method*), 92
- `fetch_auto_moderation_rules()` (*novus.BaseGuild method*), 16
- `fetch_auto_moderation_rules()` (*novus.Guild method*), 50
- `fetch_auto_moderation_rules()` (*novus.GuildPreview method*), 71
- `fetch_auto_moderation_rules()` (*novus.PartialGuild method*), 93
- `fetch_ban()` (*novus.BaseGuild method*), 25
- `fetch_ban()` (*novus.Guild method*), 50
- `fetch_ban()` (*novus.GuildPreview method*), 71
- `fetch_ban()` (*novus.PartialGuild method*), 93
- `fetch_bans()` (*novus.BaseGuild method*), 25
- `fetch_bans()` (*novus.Guild method*), 50
- `fetch_bans()` (*novus.GuildPreview method*), 71
- `fetch_bans()` (*novus.PartialGuild method*), 93
- `fetch_channels()` (*novus.BaseGuild method*), 17
- `fetch_channels()` (*novus.Guild method*), 51
- `fetch_channels()` (*novus.GuildPreview method*), 71
- `fetch_channels()` (*novus.PartialGuild method*), 93
- `fetch_emoji()` (*novus.BaseGuild method*), 18
- `fetch_emoji()` (*novus.Guild method*), 51
- `fetch_emoji()` (*novus.GuildPreview method*), 71
- `fetch_emoji()` (*novus.PartialGuild method*), 93
- `fetch_invites()` (*novus.BaseGuild method*), 15
- `fetch_invites()` (*novus.Channel method*), 29
- `fetch_invites()` (*novus.Guild method*), 51
- `fetch_invites()` (*novus.GuildPreview method*), 72
- `fetch_invites()` (*novus.PartialGuild method*), 94
- `fetch_me()` (*novus.BaseGuild method*), 22
- `fetch_me()` (*novus.Guild method*), 51
- `fetch_me()` (*novus.GuildMember class method*), 58
- `fetch_me()` (*novus.GuildPreview method*), 72
- `fetch_me()` (*novus.PartialGuild method*), 94
- `fetch_me()` (*novus.User class method*), 111
- `fetch_member()` (*novus.BaseGuild method*), 22
- `fetch_member()` (*novus.Guild method*), 52
- `fetch_member()` (*novus.GuildPreview method*), 72
- `fetch_member()` (*novus.PartialGuild method*), 94
- `fetch_members()` (*novus.BaseGuild method*), 23
- `fetch_members()` (*novus.Guild method*), 52
- `fetch_members()` (*novus.GuildPreview method*), 72
- `fetch_members()` (*novus.PartialGuild method*), 94
- `fetch_message()` (*novus.Channel method*), 30
- `fetch_messages()` (*novus.Channel method*), 29
- `fetch_my_guilds()` (*novus.User class method*), 111
- `fetch_original()` (*novus.GuildInteraction method*), 145
- `fetch_original()` (*novus.Interaction method*), 144
- `fetch_pinned_messages()` (*novus.Channel method*), 31
- `fetch_reactions()` (*novus.Message method*), 80
- `fetch_reactions()` (*novus.WebhookMessage method*), 118
- `fetch_roles()` (*novus.BaseGuild method*), 19
- `fetch_roles()` (*novus.Guild method*), 52
- `fetch_roles()` (*novus.GuildPreview method*), 73
- `fetch_roles()` (*novus.PartialGuild method*), 95
- `fetch_scheduled_events()` (*novus.BaseGuild method*), 20
- `fetch_scheduled_events()` (*novus.Guild method*), 52
- `fetch_scheduled_events()` (*novus.GuildPreview method*), 73
- `fetch_scheduled_events()` (*novus.PartialGuild method*), 95
- `fetch_sticker()` (*novus.BaseGuild method*), 21
- `fetch_sticker()` (*novus.Guild method*), 53
- `fetch_sticker()` (*novus.GuildPreview method*), 73
- `fetch_sticker()` (*novus.PartialGuild method*), 95
- `fetch_users()` (*novus.ScheduledEvent method*), 102
- `fields` (*novus.Embed attribute*), 121
- `File` (*class in novus*), 123
- `filename` (*novus.Attachment attribute*), 6
- `filename` (*novus.File attribute*), 124
- `flags` (*novus.Application attribute*), 5
- `flags` (*novus.GuildMember attribute*), 56
- `flags` (*novus.Message attribute*), 78
- `flags` (*novus.User attribute*), 110
- `flatten()` (*novus.api.APIIterator method*), 167



[follow\(\)](#) (*novus.Channel* method), 32  
[follow\\_announcement\\_channel\(\)](#)  
 (*novus.api.channel.ChannelHTTPConnection*  
*method*), 169  
[followup\(\)](#) (*novus.GuildInteraction* method), 145  
[followup\(\)](#) (*novus.Interaction* method), 143  
[footer](#) (*novus.Embed* attribute), 120  
[format\\_type](#) (*novus.Sticker* attribute), 105  
[ForumLayout](#) (class in *novus*), 155  
[ForumSortOrder](#) (class in *novus*), 155  
[ForumTag](#) (class in *novus*), 35  
[from\\_commands\(\)](#) (*novus.ext.client.CommandGroup*  
*class method*), 186  
[from\\_str\(\)](#) (*novus.Emoji* class method), 34  
[from\\_str\(\)](#) (*novus.PartialEmoji* class method), 82  
[from\\_url\(\)](#) (*novus.Webhook* class method), 114  
[FULL](#) (*novus.VideoQualityMode* attribute), 155  
[func](#) (*novus.ext.client.Loop* attribute), 188

## G

[GALLERY\\_VIEW](#) (*novus.ForumLayout* attribute), 155  
[GAME](#) (*novus.ActivityType* attribute), 159  
[gateway\\_guild\\_members](#) (*novus.ApplicationFlags* at-  
*tribute*), 160  
[gateway\\_guild\\_members\\_limited](#)  
 (*novus.ApplicationFlags* attribute), 160  
[gateway\\_message\\_content](#) (*novus.ApplicationFlags*  
*attribute*), 160  
[gateway\\_message\\_content\\_limited](#)  
 (*novus.ApplicationFlags* attribute), 160  
[gateway\\_presence](#) (*novus.ApplicationFlags* attribute),  
 160  
[gateway\\_presence\\_limited](#) (*novus.ApplicationFlags*  
*attribute*), 160  
[GatewayOpcode](#) (class in *novus*), 155  
[get\\_active\\_guild\\_threads\(\)](#)  
 (*novus.api.guild.GuildHTTPConnection*  
*method*), 171  
[get\\_application\\_role\\_records\(\)](#)  
 (*novus.api.application\_role\_connection\_metadata.ApplicationRoleHTTPConnection*  
*method*), 167  
[get\\_auto\\_moderation\\_rule\(\)](#)  
 (*novus.api.auto\_moderation.AutoModerationHTTPConnection*  
*method*), 168  
[get\\_channel\(\)](#) (*novus.api.channel.ChannelHTTPConnection*  
*method*), 168  
[get\\_channel\(\)](#) (*novus.ext.client.Client* method), 179  
[get\\_channel\(\)](#) (*novus.Guild* method), 41  
[get\\_channel\\_invites\(\)](#)  
 (*novus.api.channel.ChannelHTTPConnection*  
*method*), 169  
[get\\_channel\\_message\(\)](#)  
 (*novus.api.channel.ChannelHTTPConnection*  
*method*), 168

[get\\_channel\\_messages\(\)](#)  
 (*novus.api.channel.ChannelHTTPConnection*  
*method*), 168  
[get\\_channel\\_webhooks\(\)](#)  
 (*novus.api.webhook.WebhookHTTPConnection*  
*method*), 174  
[get\\_command\(\)](#) (*novus.ext.client.Client* method), 180  
[get\\_current\\_user\(\)](#) (*novus.api.user.UserHTTPConnection*  
*method*), 173  
[get\\_current\\_user\\_guild\\_member\(\)](#)  
 (*novus.api.user.UserHTTPConnection*  
*method*), 174  
[get\\_current\\_user\\_guilds\(\)](#)  
 (*novus.api.user.UserHTTPConnection*  
*method*), 174  
[get\\_emoji\(\)](#) (*novus.api.emoji.EmojiHTTPConnection*  
*method*), 170  
[get\\_event\(\)](#) (*novus.Guild* method), 41  
[get\\_guild\(\)](#) (*novus.api.guild.GuildHTTPConnection*  
*method*), 171  
[get\\_guild\(\)](#) (*novus.ext.client.Client* method), 179  
[get\\_guild\\_audit\\_log\(\)](#)  
 (*novus.api.audit\_log.AuditLogHTTPConnection*  
*method*), 168  
[get\\_guild\\_ban\(\)](#) (*novus.api.guild.GuildHTTPConnection*  
*method*), 172  
[get\\_guild\\_bans\(\)](#) (*novus.api.guild.GuildHTTPConnection*  
*method*), 172  
[get\\_guild\\_channels\(\)](#)  
 (*novus.api.guild.GuildHTTPConnection*  
*method*), 171  
[get\\_guild\\_invites\(\)](#)  
 (*novus.api.guild.GuildHTTPConnection*  
*method*), 172  
[get\\_guild\\_member\(\)](#) (*novus.api.guild.GuildHTTPConnection*  
*method*), 171  
[get\\_guild\\_members\(\)](#)  
 (*novus.api.guild.GuildHTTPConnection*  
*method*), 171  
[get\\_guild\\_preview\(\)](#)  
 (*novus.api.guild.GuildHTTPConnection*  
*method*), 171  
[get\\_guild\\_roles\(\)](#) (*novus.api.guild.GuildHTTPConnection*  
*method*), 172  
[get\\_guild\\_scheduled\\_event\(\)](#)  
 (*novus.api.guild\_scheduled\_event.GuildEventHTTPConnection*  
*method*), 172  
[get\\_guild\\_scheduled\\_event\\_users\(\)](#)  
 (*novus.api.guild\_scheduled\_event.GuildEventHTTPConnection*  
*method*), 172  
[get\\_guild\\_sticker\(\)](#)  
 (*novus.api.sticker.StickerHTTPConnection*  
*method*), 173  
[get\\_guild\\_webhooks\(\)](#)

- `(novus.api.webhook.WebhookHTTPConnection method)`, 174
- `get_invite()` (*novus.api.invite.InviteHTTPConnection method*), 173
- `get_member()` (*novus.Guild method*), 40
- `get_mention()` (*novus.ext.client.Command method*), 184
- `get_mention()` (*novus.ext.client.CommandGroup method*), 187
- `get_pinned_messages()` (*novus.api.channel.ChannelHTTPConnection method*), 169
- `get_plugin()` (*novus.ext.client.Client method*), 180
- `get_reactions()` (*novus.api.channel.ChannelHTTPConnection method*), 169
- `get_role()` (*novus.Guild method*), 41
- `get_stage_instance()` (*novus.api.stage\_instance.StageHTTPConnection method*), 173
- `get_sticker()` (*novus.api.sticker.StickerHTTPConnection method*), 173
- `get_sticker()` (*novus.Guild method*), 40
- `get_thread()` (*novus.Guild method*), 41
- `get_thread_member()` (*novus.api.channel.ChannelHTTPConnection method*), 170
- `get_url()` (*novus.Asset method*), 6
- `get_user()` (*novus.api.user.UserHTTPConnection method*), 173
- `get_user()` (*novus.ext.client.Client method*), 179
- `get_user_application_role_connection()` (*novus.api.user.UserHTTPConnection method*), 174
- `get_user_connections()` (*novus.api.user.UserHTTPConnection method*), 174
- `get_webhook()` (*novus.api.webhook.WebhookHTTPConnection method*), 174
- `get_webhook_message()` (*novus.api.webhook.WebhookHTTPConnection method*), 174
- GIF (*novus.StickerFormat attribute*), 157
- global\_name (*novus.GuildMember attribute*), 55
- global\_name (*novus.User attribute*), 109
- GRAY (*novus.ButtonStyle attribute*), 153
- GREEN (*novus.ButtonStyle attribute*), 153
- GREY (*novus.ButtonStyle attribute*), 153
- GROUP\_DM (*novus.ChannelType attribute*), 153
- Guild (*class in novus*), 36
- guild (*novus.api.HTTPConnection attribute*), 166
- guild (*novus.ApplicationCommandData attribute*), 148
- guild (*novus.AutoModerationRule attribute*), 11
- guild (*novus.Channel attribute*), 26
- guild (*novus.ContextComandData attribute*), 148
- guild (*novus.Emoji attribute*), 33
- guild (*novus.GuildMember attribute*), 57
- guild (*novus.Interaction attribute*), 142
- guild (*novus.Invite attribute*), 75
- guild (*novus.Message attribute*), 76
- guild (*novus.Role attribute*), 98
- guild (*novus.ScheduledEvent attribute*), 99
- guild (*novus.Sticker attribute*), 105
- GUILD (*novus.StickerType attribute*), 158
- guild (*novus.VoiceState attribute*), 112
- GUILD\_ANNOUNCEMENT (*novus.ChannelType attribute*), 154
- GUILD\_APPLICATION\_PREMIUM\_SUBSCRIPTION (*novus.MessageType attribute*), 156
- guild\_avatar (*novus.GuildMember attribute*), 56
- guild\_avatar\_hash (*novus.GuildMember attribute*), 56
- GUILD\_BOOST (*novus.MessageType attribute*), 156
- GUILD\_BOOST\_TIER\_1 (*novus.MessageType attribute*), 156
- GUILD\_BOOST\_TIER\_2 (*novus.MessageType attribute*), 156
- GUILD\_BOOST\_TIER\_3 (*novus.MessageType attribute*), 156
- GUILD\_CATEGORY (*novus.ChannelType attribute*), 154
- GUILD\_DIRECTORY (*novus.ChannelType attribute*), 154
- GUILD\_DISCOVERY\_DISQUALIFIED (*novus.MessageType attribute*), 156
- GUILD\_DISCOVERY\_GRACE\_PERIOD\_FINAL\_WARNING (*novus.MessageType attribute*), 156
- GUILD\_DISCOVERY\_GRACE\_PERIOD\_INITIAL\_WARNING (*novus.MessageType attribute*), 156
- GUILD\_DISCOVERY\_REQUALIFIED (*novus.MessageType attribute*), 156
- guild\_emojis\_and\_stickers (*novus.Intents attribute*), 159
- GUILD\_FORUM (*novus.ChannelType attribute*), 154
- guild\_id (*novus.Application attribute*), 5
- guild\_id (*novus.ApplicationCommand attribute*), 137
- guild\_id (*novus.AutoModerationRule attribute*), 9
- guild\_id (*novus.StageInstance attribute*), 103
- guild\_id (*novus.Webhook attribute*), 113
- guild\_ids (*novus.ext.client.Command attribute*), 184
- guild\_ids (*novus.ext.client.CommandGroup attribute*), 186
- guild\_integrations (*novus.Intents attribute*), 159
- GUILD\_INVITE\_REMINDER (*novus.MessageType attribute*), 157
- guild\_invites (*novus.Intents attribute*), 159
- guild\_locale (*novus.Interaction attribute*), 142
- guild\_members (*novus.Intents attribute*), 159
- guild\_message\_reactions (*novus.Intents attribute*), 159
- guild\_message\_typing (*novus.Intents attribute*), 159
- guild\_messages (*novus.Intents attribute*), 159

- guild\_moderation (novus.Intent attribute), 159  
 GUILD\_ONLY (novus.EventPrivacyLevel attribute), 154  
 guild\_presences (novus.Intent attribute), 159  
 guild\_scheduled\_event (novus.api.HTTPConnection attribute), 166  
 guild\_scheduled\_event\_create (novus.AuditLogEntryType attribute), 164  
 GUILD\_SCHEDULED\_EVENT\_CREATE (novus.AuditLogEventType attribute), 151  
 guild\_scheduled\_event\_delete (novus.AuditLogEntryType attribute), 164  
 GUILD\_SCHEDULED\_EVENT\_DELETE (novus.AuditLogEventType attribute), 151  
 guild\_scheduled\_event\_update (novus.AuditLogEntryType attribute), 164  
 GUILD\_SCHEDULED\_EVENT\_UPDATE (novus.AuditLogEventType attribute), 151  
 guild\_scheduled\_events (novus.Intent attribute), 159  
 GUILD\_STAGE\_VOICE (novus.ChannelType attribute), 154  
 GUILD\_SYNC (novus.GatewayOpcode attribute), 155  
 guild\_template (novus.api.HTTPConnection attribute), 167  
 GUILD\_TEXT (novus.ChannelType attribute), 154  
 guild\_update (novus.AuditLogEntryType attribute), 162  
 GUILD\_UPDATE (novus.AuditLogEventType attribute), 151  
 GUILD\_VOICE (novus.ChannelType attribute), 154  
 guild\_voice\_states (novus.Intent attribute), 159  
 guild\_webhooks (novus.Intent attribute), 159  
 GuildBan (class in novus), 54  
 GuildEventHTTPConnection (class in novus.api.guild\_scheduled\_event), 172  
 GuildHTTPConnection (class in novus.api.guild), 171  
 GuildInteraction (class in novus), 144  
 GuildMember (class in novus), 54  
 GuildPreview (class in novus), 60  
 guilds (novus.Intent attribute), 159  
 GuildTemplateHTTPConnection (class in novus.api.guild\_template), 173
- ## H
- HEARTBEAT (novus.GatewayOpcode attribute), 155  
 HEARTBEAT\_ACK (novus.GatewayOpcode attribute), 155  
 HELLO (novus.GatewayOpcode attribute), 155  
 HIGH (novus.VerificationLevel attribute), 158  
 hoist (novus.Role attribute), 97  
 HTTPConnection (class in novus.api), 166  
 hypesquad (novus.UserFlags attribute), 162  
 hypesquad\_house\_balance (novus.UserFlags attribute), 162  
 hypesquad\_house\_bravery (novus.UserFlags attribute), 162  
 hypesquad\_house\_brilliance (novus.UserFlags attribute), 162
- ## I
- icon (novus.Application attribute), 4  
 icon (novus.Guild attribute), 36  
 icon (novus.GuildPreview attribute), 61  
 icon (novus.PartialGuild attribute), 83  
 icon (novus.Role attribute), 97  
 icon (novus.Team attribute), 107  
 icon\_hash (novus.Application attribute), 4  
 icon\_hash (novus.Guild attribute), 36  
 icon\_hash (novus.GuildPreview attribute), 61  
 icon\_hash (novus.PartialGuild attribute), 83  
 icon\_hash (novus.Role attribute), 97  
 icon\_hash (novus.Team attribute), 107  
 id (novus.Application attribute), 4  
 id (novus.ApplicationCommand attribute), 136  
 id (novus.ApplicationCommandData attribute), 148  
 id (novus.Attachment attribute), 6  
 id (novus.AuditLogEntry attribute), 8  
 id (novus.AutoModerationRule attribute), 9  
 id (novus.BaseGuild attribute), 13  
 id (novus.Channel attribute), 26  
 id (novus.ContextComandData attribute), 147  
 id (novus.Emoji attribute), 33  
 id (novus.ForumTag attribute), 35  
 id (novus.Guild attribute), 36  
 id (novus.GuildMember attribute), 54  
 id (novus.GuildPreview attribute), 60  
 id (novus.Interaction attribute), 141  
 id (novus.Message attribute), 76  
 id (novus.MessageInteraction attribute), 147  
 id (novus.PartialEmoji attribute), 82  
 id (novus.PartialGuild attribute), 82  
 id (novus.PermissionOverwrite attribute), 124  
 id (novus.Role attribute), 97  
 id (novus.ScheduledEvent attribute), 99  
 id (novus.StageInstance attribute), 103  
 id (novus.Sticker attribute), 104  
 id (novus.Team attribute), 107  
 id (novus.ThreadMember attribute), 108  
 id (novus.User attribute), 109  
 id (novus.Webhook attribute), 113  
 IDENTIFY (novus.GatewayOpcode attribute), 155  
 IDLE (novus.Status attribute), 158  
 image (novus.Embed attribute), 120  
 image (novus.ScheduledEvent attribute), 100  
 image\_hash (novus.ScheduledEvent attribute), 100  
 immediate (novus.ext.client.LoopBehavior attribute), 189  
 insert\_field\_at() (novus.Embed method), 123

install\_permissions (*novus.Application* attribute), 6  
 install\_scopes (*novus.Application* attribute), 6  
 INTEGER (*novus.ApplicationOptionType* attribute), 150  
 integration\_create (*novus.AuditLogEntryType* attribute), 163  
 INTEGRATION\_CREATE (*novus.AuditLogEventType* attribute), 151  
 integration\_delete (*novus.AuditLogEntryType* attribute), 163  
 INTEGRATION\_DELETE (*novus.AuditLogEventType* attribute), 151  
 integration\_update (*novus.AuditLogEntryType* attribute), 163  
 INTEGRATION\_UPDATE (*novus.AuditLogEventType* attribute), 151  
 Intents (class in *novus*), 159  
 InteractableComponent (class in *novus*), 150  
 Interaction (class in *novus*), 141  
 interaction (*novus.api.HTTPConnection* attribute), 167  
 interaction (*novus.Message* attribute), 78  
 INTERACTION\_PREMIUM\_UPSELL (*novus.MessageType* attribute), 157  
 InteractionData (class in *novus*), 149  
 InteractionOption (class in *novus*), 149  
 InteractionResolved (class in *novus*), 149  
 InteractionResponseType (class in *novus*), 155  
 InteractionType (class in *novus*), 156  
 INVALIDATE\_SESSION (*novus.GatewayOpcode* attribute), 155  
 INVISIBLE (*novus.Status* attribute), 158  
 Invite (class in *novus*), 74  
 invite (*novus.api.HTTPConnection* attribute), 167  
 invite\_create (*novus.AuditLogEntryType* attribute), 163  
 INVITE\_CREATE (*novus.AuditLogEventType* attribute), 152  
 invite\_delete (*novus.AuditLogEntryType* attribute), 163  
 INVITE\_DELETE (*novus.AuditLogEventType* attribute), 152  
 invite\_update (*novus.AuditLogEntryType* attribute), 163  
 INVITE\_UPDATE (*novus.AuditLogEventType* attribute), 152  
 InviteHTTPConnection (class in *novus.api.invite*), 173  
 is\_subcommand (*novus.ext.client.Command* attribute), 184

## J

join\_thread() (*novus.Channel* method), 32  
 join\_timestamp (*novus.ThreadMember* attribute), 108  
 joined\_at (*novus.GuildMember* attribute), 57  
 jump\_url (*novus.Message* attribute), 79

## K

KEYWORD (*novus.AutoModerationTriggerType* attribute), 153  
 KEYWORD\_PRESET (*novus.AutoModerationTriggerType* attribute), 153  
 kick() (*novus.BaseGuild* method), 25  
 kick() (*novus.Guild* method), 53  
 kick() (*novus.GuildMember* method), 60  
 kick() (*novus.GuildPreview* method), 73  
 kick() (*novus.PartialGuild* method), 95  
 kick\_members (*novus.Permissions* attribute), 160  
 kwargs (*novus.api.\_route.Route* attribute), 166

## L

label (*novus.Button* attribute), 127  
 label (*novus.SelectOption* attribute), 133  
 label (*novus.TextInput* attribute), 134  
 LATEST\_ACTIVITY (*novus.ForumSortOrder* attribute), 155  
 LayoutComponent (class in *novus*), 150  
 leave() (*novus.BaseGuild* method), 22  
 leave() (*novus.Guild* method), 53  
 leave() (*novus.GuildPreview* method), 73  
 leave() (*novus.PartialGuild* method), 95  
 leave\_guild() (*novus.api.user.UserHTTPConnection* method), 174  
 leave\_thread() (*novus.Channel* method), 32  
 LINK (*novus.ButtonStyle* attribute), 153  
 list\_auto\_moderation\_rules\_for\_guild() (*novus.api.auto\_moderation.AutoModerationHTTPConnection* method), 168  
 list\_guild\_emojis() (*novus.api.emoji.EmojiHTTPConnection* method), 170  
 list\_guild\_stickers() (*novus.api.sticker.StickerHTTPConnection* method), 173  
 list\_joined\_private\_archived\_threads() (*novus.api.channel.ChannelHTTPConnection* method), 170  
 list\_private\_archived\_threads() (*novus.api.channel.ChannelHTTPConnection* method), 170  
 list\_public\_archived\_threads() (*novus.api.channel.ChannelHTTPConnection* method), 170  
 list\_scheduled\_events\_for\_guild() (*novus.api.guild\_scheduled\_event.GuildEventHTTPConnection* method), 172  
 list\_thread\_members() (*novus.api.channel.ChannelHTTPConnection* method), 170  
 LIST\_VIEW (*novus.ForumLayout* attribute), 155  
 LISTENING (*novus.ActivityType* attribute), 159



- load\_plugins() (*novus.ext.client.Client* method), 180
- locale (*novus.GuildMember* attribute), 56
- locale (*novus.Interaction* attribute), 142
- locale (*novus.User* attribute), 110
- location (*novus.ScheduledEvent* attribute), 100
- LONG (*novus.TextInputStyle* attribute), 158
- LONG\_DATE (*novus.TimestampFormat* attribute), 158
- LONG\_DATETIME (*novus.TimestampFormat* attribute), 158
- LONG\_TIME (*novus.TimestampFormat* attribute), 158
- Loop (*class in novus.ext.client*), 188
- loop() (*in module novus.ext.client*), 188
- loop\_time (*novus.ext.client.Loop* attribute), 188
- LoopBehavior (*class in novus.ext.client*), 189
- LOTTIE (*novus.StickerFormat* attribute), 157
- LOW (*novus.VerificationLevel* attribute), 158
- ## M
- manage\_channels (*novus.Permissions* attribute), 161
- manage\_emojis\_and\_stickers (*novus.Permissions* attribute), 161
- manage\_events (*novus.Permissions* attribute), 162
- manage\_guild (*novus.Permissions* attribute), 161
- manage\_messages (*novus.Permissions* attribute), 161
- manage\_nicknames (*novus.Permissions* attribute), 161
- manage\_roles (*novus.Permissions* attribute), 161
- manage\_threads (*novus.Permissions* attribute), 162
- manage\_webhooks (*novus.Permissions* attribute), 161
- managed (*novus.Emoji* attribute), 33
- managed (*novus.Role* attribute), 98
- max\_age (*novus.Invite* attribute), 75
- max\_length (*novus.ApplicationCommandOption* attribute), 141
- max\_length (*novus.TextInput* attribute), 134
- max\_members (*novus.Guild* attribute), 38
- max\_presences (*novus.Guild* attribute), 38
- max\_uses (*novus.Invite* attribute), 75
- max\_value (*novus.ApplicationCommandOption* attribute), 141
- max\_values (*novus.ChannelSelectMenu* attribute), 130
- max\_values (*novus.MentionableSelectMenu* attribute), 133
- max\_values (*novus.RoleSelectMenu* attribute), 131
- max\_values (*novus.StringSelectMenu* attribute), 128
- max\_values (*novus.UserSelectMenu* attribute), 132
- max\_video\_channel\_users (*novus.Guild* attribute), 39
- me (*novus.ext.client.Client* attribute), 178
- MEDIUM (*novus.VerificationLevel* attribute), 158
- MEMBER (*novus.PermissionOverwriteType* attribute), 157
- member (*novus.ThreadMember* attribute), 108
- member\_ban\_add (*novus.AuditLogEntryType* attribute), 163
- MEMBER\_BAN\_ADD (*novus.AuditLogEventType* attribute), 152
- member\_ban\_remove (*novus.AuditLogEntryType* attribute), 163
- MEMBER\_BAN\_REMOVE (*novus.AuditLogEventType* attribute), 152
- member\_disconnect (*novus.AuditLogEntryType* attribute), 163
- MEMBER\_DISCONNECT (*novus.AuditLogEventType* attribute), 152
- member\_kick (*novus.AuditLogEntryType* attribute), 163
- MEMBER\_KICK (*novus.AuditLogEventType* attribute), 152
- member\_move (*novus.AuditLogEntryType* attribute), 163
- MEMBER\_MOVE (*novus.AuditLogEventType* attribute), 152
- member\_prune (*novus.AuditLogEntryType* attribute), 163
- MEMBER\_PRUNE (*novus.AuditLogEventType* attribute), 152
- member\_role\_update (*novus.AuditLogEntryType* attribute), 163
- MEMBER\_ROLE\_UPDATE (*novus.AuditLogEventType* attribute), 152
- member\_update (*novus.AuditLogEntryType* attribute), 163
- MEMBER\_UPDATE (*novus.AuditLogEventType* attribute), 152
- members (*novus.InteractionResolved* attribute), 149
- members (*novus.Team* attribute), 107
- MEMBERS\_WITHOUT\_ROLES (*novus.ContentFilterLevel* attribute), 154
- MENTIONABLE (*novus.ApplicationOptionType* attribute), 150
- mention (*novus.GuildMember* property), 57
- mention (*novus.User* property), 111
- mention\_channels (*novus.Message* attribute), 77
- mention\_everyone (*novus.Message* attribute), 76
- mention\_everyone (*novus.Permissions* attribute), 161
- mention\_roles (*novus.Message* attribute), 77
- MENTION\_SPAM (*novus.AutoModerationTriggerType* attribute), 153
- mentionable (*novus.Role* attribute), 98
- MENTIONABLE\_SELECT (*novus.ComponentType* attribute), 154
- MentionableSelectMenu (*class in novus*), 132
- mentions (*novus.Message* attribute), 76
- Message (*class in novus*), 75
- MESSAGE (*novus.ApplicationCommandType* attribute), 150
- message (*novus.Interaction* attribute), 142
- message (*novus.Reaction* attribute), 96
- message\_bulk\_delete (*novus.AuditLogEntryType* attribute), 163
- MESSAGE\_BULK\_DELETE (*novus.AuditLogEventType* attribute), 152
- MESSAGE\_COMPONENT (*novus.InteractionType* attribute), 156

- message\_content (*novus.Intent* attribute), 159
  - message\_delete (*novus.AuditLogEntryType* attribute), 163
  - MESSAGE\_DELETE (*novus.AuditLogEventType* attribute), 152
  - message\_pin (*novus.AuditLogEntryType* attribute), 163
  - MESSAGE\_PIN (*novus.AuditLogEventType* attribute), 152
  - message\_reference (*novus.Message* attribute), 78
  - MESSAGE\_SEND (*novus.AutoModerationEventType* attribute), 153
  - message\_unpin (*novus.AuditLogEntryType* attribute), 163
  - MESSAGE\_UNPIN (*novus.AuditLogEventType* attribute), 152
  - MessageComponentData (class in *novus*), 149
  - MessageFlags (class in *novus*), 160
  - MessageInteraction (class in *novus*), 147
  - messages (*novus.InteractionResolved* attribute), 150
  - messages() (*novus.Channel* method), 30
  - MessageType (class in *novus*), 156
  - method (*novus.api.\_route.Route* attribute), 165
  - mfa\_enabled (*novus.GuildMember* attribute), 55
  - mfa\_enabled (*novus.User* attribute), 109
  - mfa\_level (*novus.Guild* attribute), 38
  - MFALevel (class in *novus*), 156
  - min\_length (*novus.ApplicationCommandOption* attribute), 141
  - min\_length (*novus.TextInput* attribute), 134
  - min\_value (*novus.ApplicationCommandOption* attribute), 140
  - min\_values (*novus.ChannelSelectMenu* attribute), 130
  - min\_values (*novus.MentionableSelectMenu* attribute), 132
  - min\_values (*novus.RoleSelectMenu* attribute), 131
  - min\_values (*novus.StringSelectMenu* attribute), 128
  - min\_values (*novus.UserSelectMenu* attribute), 132
  - MODAL (*novus.InteractionResponseType* attribute), 156
  - MODAL\_SUBMIT (*novus.InteractionType* attribute), 156
  - ModalSubmitData (class in *novus*), 149
  - moderate\_members (*novus.Permissions* attribute), 162
  - moderated (*novus.ForumTag* attribute), 36
  - modify\_auto\_moderation\_rule() (*novus.api.auto\_moderation.AutoModerationHTTPConnection* method), 168
  - modify\_channel() (*novus.api.channel.ChannelHTTPConnection* method), 168
  - modify\_current\_user() (*novus.api.user.UserHTTPConnection* method), 174
  - modify\_guild() (*novus.api.guild.GuildHTTPConnection* method), 171
  - modify\_guild\_emoji() (*novus.api.emoji.EmojiHTTPConnection* method), 170
  - modify\_guild\_member() (*novus.api.guild.GuildHTTPConnection* method), 171
  - modify\_guild\_role() (*novus.api.guild.GuildHTTPConnection* method), 172
  - modify\_guild\_scheduled\_event() (*novus.api.guild\_scheduled\_event.GuildEventHTTPConnection* method), 172
  - modify\_guild\_sticker() (*novus.api.sticker.StickerHTTPConnection* method), 173
  - modify\_stage\_instance() (*novus.api.stage\_instance.StageHTTPConnection* method), 173
  - modify\_webhook() (*novus.api.webhook.WebhookHTTPConnection* method), 174
  - move\_members (*novus.Permissions* attribute), 161
  - mute (*novus.GuildMember* attribute), 57
  - mute (*novus.VoiceState* attribute), 113
  - mute\_members (*novus.Permissions* attribute), 161
- ## N
- name (*novus.Activity* attribute), 3
  - name (*novus.Application* attribute), 4
  - name (*novus.ApplicationCommand* attribute), 137
  - name (*novus.ApplicationCommandChoice* attribute), 138
  - name (*novus.ApplicationCommandData* attribute), 148
  - name (*novus.ApplicationCommandOption* attribute), 139
  - name (*novus.AutoModerationRule* attribute), 10
  - name (*novus.BaseGuild* attribute), 13
  - name (*novus.ContextComandData* attribute), 147
  - name (*novus.Emoji* attribute), 33
  - name (*novus.ext.client.Command* attribute), 183
  - name (*novus.ext.client.CommandGroup* attribute), 185
  - name (*novus.ForumTag* attribute), 35
  - name (*novus.Guild* attribute), 36
  - name (*novus.GuildPreview* attribute), 61
  - name (*novus.MessageInteraction* attribute), 147
  - name (*novus.PartialApplicationCommand* attribute), 135
  - name (*novus.PartialEmoji* attribute), 82
  - name (*novus.PartialGuild* attribute), 82
  - name (*novus.PartialRole* attribute), 97
  - name (*novus.ScheduledEvent* attribute), 99
  - name (*novus.Sticker* attribute), 105
  - name (*novus.Team* attribute), 107
  - name (*novus.Webhook* attribute), 114
  - name\_localizations (*novus.ApplicationCommand* attribute), 137
  - name\_localizations (*novus.ApplicationCommandChoice* attribute), 138
  - name\_localizations (*novus.ApplicationCommandOption* attribute), 140

- name\_localizations (novus.PartialApplicationCommand attribute), 136
- nick (novus.GuildMember attribute), 56
- NITRO (novus.UserPremiumType attribute), 158
- NITRO\_BASIC (novus.UserPremiumType attribute), 158
- NITRO\_CLASSIC (novus.UserPremiumType attribute), 158
- NONE (novus.MFALevel attribute), 156
- NONE (novus.PremiumTier attribute), 157
- NONE (novus.UserPremiumType attribute), 158
- NONE (novus.VerificationLevel attribute), 158
- none() (novus.AllowedMentions class method), 119
- NOT\_SET (novus.ForumLayout attribute), 155
- NotificationLevel (class in novus), 157
- nsfw (novus.ApplicationCommand attribute), 138
- nsfw (novus.PartialApplicationCommand attribute), 136
- nsfw\_level (novus.Guild attribute), 40
- nsfw\_level (novus.PartialGuild attribute), 84
- NSFWLevel (class in novus), 157
- NUMBER (novus.ApplicationOptionType attribute), 151
- ## O
- oauth2 (novus.api.HTTPConnection attribute), 167
- Object (class in novus), 124
- OFFLINE (novus.Status attribute), 159
- on\_load() (novus.ext.client.Plugin method), 182
- on\_unload() (novus.ext.client.Plugin method), 182
- ONLINE (novus.Status attribute), 159
- only() (novus.AllowedMentions class method), 119
- ONLY\_MENTIONS (novus.NotificationLevel attribute), 157
- options (novus.ApplicationCommand attribute), 137
- options (novus.ApplicationCommandData attribute), 148
- options (novus.ApplicationCommandOption attribute), 140
- options (novus.AuditLogEntry attribute), 8
- options (novus.ContextComandData attribute), 148
- options (novus.PartialApplicationCommand attribute), 136
- options (novus.StringSelectMenu attribute), 128
- owner (novus.Application attribute), 5
- owner (novus.ext.client.Loop attribute), 189
- owner\_id (novus.Guild attribute), 37
- owner\_user\_id (novus.Team attribute), 108
- ## P
- pack\_id (novus.Sticker attribute), 104
- PARAGRAPH (novus.TextInputStyle attribute), 158
- partial() (novus.Channel class method), 26
- partial() (novus.Webhook class method), 114
- PartialApplicationCommand (class in novus), 135
- PartialEmoji (class in novus), 81
- PartialGuild (class in novus), 82
- partner (novus.UserFlags attribute), 162
- path (novus.api.\_route.Route attribute), 165
- pending (novus.GuildMember attribute), 57
- PermissionOverwrite (class in novus), 124
- PermissionOverwriteType (class in novus), 157
- Permissions (class in novus), 160
- permissions (novus.GuildMember attribute), 57
- permissions (novus.GuildMember property), 58
- permissions (novus.Role attribute), 98
- permissions (novus.TeamMember attribute), 108
- permissions\_for() (novus.Channel method), 27
- permissions\_in() (novus.GuildMember method), 58
- pin() (novus.Message method), 81
- pin() (novus.WebhookMessage method), 118
- pin\_message() (novus.api.channel.ChannelHTTPConnection method), 169
- PING (novus.InteractionType attribute), 156
- pinned (novus.Message attribute), 77
- placeholder (novus.ChannelSelectMenu attribute), 130
- placeholder (novus.MentionableSelectMenu attribute), 132
- placeholder (novus.RoleSelectMenu attribute), 131
- placeholder (novus.StringSelectMenu attribute), 128
- placeholder (novus.TextInput attribute), 135
- placeholder (novus.UserSelectMenu attribute), 132
- Plugin (class in novus.ext.client), 182
- plugins (novus.ext.client.Client attribute), 178
- PNG (novus.StickerFormat attribute), 157
- PONG (novus.InteractionResponseType attribute), 156
- pong() (novus.GuildInteraction method), 145
- pong() (novus.Interaction method), 142
- pop() (novus.ActionRow method), 126
- pop() (novus.StringSelectMenu method), 129
- position (novus.Message attribute), 79
- position (novus.Role attribute), 97
- preferred\_locale (novus.Guild attribute), 39
- premium\_early\_supporter (novus.UserFlags attribute), 162
- premium\_progress\_bar\_enabled (novus.Guild attribute), 40
- premium\_since (novus.GuildMember attribute), 57
- premium\_subscription\_count (novus.Guild attribute), 39
- premium\_subscription\_count (novus.PartialGuild attribute), 84
- premium\_tier (novus.Guild attribute), 39
- premium\_type (novus.GuildMember attribute), 56
- premium\_type (novus.User attribute), 110
- PremiumTier (class in novus), 157
- PRESENCE (novus.GatewayOpcode attribute), 155
- PRIMARY (novus.ButtonStyle attribute), 153
- primary\_sku\_id (novus.Application attribute), 5
- priority\_speaker (novus.Permissions attribute), 161
- privacy\_level (novus.ScheduledEvent attribute), 99
- privacy\_level (novus.StageInstance attribute), 103

privacy\_policy\_url (*novus.Application* attribute), 5  
 PRIVATE\_THREAD (*novus.ChannelType* attribute), 154  
 PROFANITY (*novus.AutoModerationKeywordPresetType* attribute), 153  
 provider (*novus.Embed* attribute), 121  
 proxy\_url (*novus.Attachment* attribute), 7  
 PUBLIC\_THREAD (*novus.ChannelType* attribute), 154  
 public\_updates\_channel\_id (*novus.Guild* attribute), 39

## R

Reaction (*class in novus*), 96  
 reactions (*novus.Message* attribute), 77  
 read\_message\_history (*novus.Permissions* attribute), 161  
 reason (*novus.AuditLogEntry* attribute), 8  
 reason (*novus.GuildBan* attribute), 54  
 RECIPIENT\_ADD (*novus.MessageType* attribute), 157  
 RECIPIENT\_REMOVE (*novus.MessageType* attribute), 157  
 RECONNECT (*novus.GatewayOpcode* attribute), 155  
 RED (*novus.ButtonStyle* attribute), 153  
 referenced\_message (*novus.Message* attribute), 78  
 RELATIVE (*novus.TimestampFormat* attribute), 158  
 remove\_author() (*novus.Embed* method), 123  
 remove\_command() (*novus.ext.client.Client* method), 180  
 remove\_field() (*novus.Embed* method), 123  
 remove\_footer() (*novus.Embed* method), 122  
 remove\_guild\_ban() (*novus.api.guild.GuildHTTPConnection* method), 172  
 remove\_guild\_member()  
   (*novus.api.guild.GuildHTTPConnection* method), 171  
 remove\_guild\_member\_role()  
   (*novus.api.guild.GuildHTTPConnection* method), 171  
 remove\_image() (*novus.Embed* method), 122  
 remove\_member\_role() (*novus.BaseGuild* method), 25  
 remove\_member\_role() (*novus.Guild* method), 53  
 remove\_member\_role() (*novus.GuildPreview* method), 73  
 remove\_member\_role() (*novus.PartialGuild* method), 95  
 remove\_overwrite() (*novus.Channel* method), 29  
 remove\_plugin() (*novus.ext.client.Client* method), 180  
 remove\_plugin\_file() (*novus.ext.client.Client* method), 181  
 remove\_reaction() (*novus.Message* method), 80  
 remove\_reaction() (*novus.WebhookMessage* method), 118  
 remove\_role() (*novus.GuildMember* method), 60  
 remove\_thread\_member()  
   (*novus.api.channel.ChannelHTTPConnection* method), 170  
 remove\_thread\_member() (*novus.Channel* method), 32  
 remove\_thumbnail() (*novus.Embed* method), 122  
 REPLY (*novus.MessageType* attribute), 157  
 REQUEST\_MEMBERS (*novus.GatewayOpcode* attribute), 155  
 request\_to\_speak (*novus.Permissions* attribute), 161  
 request\_to\_speak\_timestamp (*novus.VoiceState* attribute), 113  
 required (*novus.ApplicationCommandOption* attribute), 140  
 required (*novus.TextInput* attribute), 134  
 requires\_colons (*novus.Emoji* attribute), 33  
 resolved (*novus.ApplicationCommandData* attribute), 148  
 resolved (*novus.ContextComandData* attribute), 147  
 resource (*novus.api.\_route.Route* attribute), 165  
 resource (*novus.Asset* attribute), 6  
 RESUME (*novus.GatewayOpcode* attribute), 155  
 Role (*class in novus*), 97  
 ROLE (*novus.ApplicationOptionType* attribute), 151  
 ROLE (*novus.PermissionOverwriteType* attribute), 157  
 role\_connections\_verification\_url  
   (*novus.Application* attribute), 6  
 role\_create (*novus.AuditLogEntryType* attribute), 163  
 ROLE\_CREATE (*novus.AuditLogEventType* attribute), 152  
 role\_delete (*novus.AuditLogEntryType* attribute), 163  
 ROLE\_DELETE (*novus.AuditLogEventType* attribute), 152  
 role\_ids (*novus.Emoji* attribute), 33  
 role\_ids (*novus.GuildMember* attribute), 56  
 ROLE\_SELECT (*novus.ComponentType* attribute), 154  
 role\_subscription\_data (*novus.Message* attribute), 79  
 ROLE\_SUBSCRIPTION\_PURCHASE (*novus.MessageType* attribute), 157  
 role\_update (*novus.AuditLogEntryType* attribute), 163  
 ROLE\_UPDATE (*novus.AuditLogEventType* attribute), 152  
 roles (*novus.Guild* attribute), 37  
 roles (*novus.InteractionResolved* attribute), 149  
 RoleSelectMenu (*class in novus*), 130  
 Route (*class in novus.api.\_route*), 165  
 rpc\_origins (*novus.Application* attribute), 4  
 rules\_channel\_id (*novus.Guild* attribute), 38  
 run() (*novus.ext.client.Client* method), 181  
 run() (*novus.ext.client.Command* method), 184  
 run() (*novus.ext.client.CommandGroup* method), 186  
 run\_autocomplete() (*novus.ext.client.Command* method), 185  
 run\_autocomplete() (*novus.ext.client.CommandGroup* method), 186  
 run\_webserver() (*novus.ext.client.Client* method), 181

## S

SAFE (*novus.NSFWLevel* attribute), 157  
 SCHEDULED (*novus.EventStatus* attribute), 155



- ScheduledEvent (class in novus), 99
- search\_guild\_members() (novus.api.guild.GuildHTTPConnection method), 171
- search\_members() (novus.BaseGuild method), 23
- search\_members() (novus.Guild method), 53
- search\_members() (novus.GuildPreview method), 74
- search\_members() (novus.PartialGuild method), 96
- SECONDARY (novus.ButtonStyle attribute), 153
- SelectOption (class in novus), 133
- self\_deaf (novus.VoiceState attribute), 113
- self\_mute (novus.VoiceState attribute), 113
- self\_video (novus.VoiceState attribute), 113
- send() (novus.Channel method), 32
- send() (novus.GuildInteraction method), 145
- send() (novus.GuildMember method), 59
- send() (novus.Interaction method), 142
- send() (novus.User method), 112
- send() (novus.Webhook method), 115
- SEND\_ALERT\_MESSAGE (novus.AutoModerationActionType attribute), 153
- send\_autocomplete() (novus.GuildInteraction method), 146
- send\_autocomplete() (novus.Interaction method), 144
- send\_messages (novus.Permissions attribute), 161
- send\_messages\_in\_threads (novus.Permissions attribute), 162
- send\_modal() (novus.GuildInteraction method), 146
- send\_modal() (novus.Interaction method), 144
- send\_tts\_messages (novus.Permissions attribute), 161
- session\_id (novus.VoiceState attribute), 113
- set() (novus.ActionRow method), 125
- set() (novus.StringSelectMenu method), 129
- set\_author() (novus.Embed method), 123
- set\_author\_from\_user() (novus.Embed method), 123
- set\_footer() (novus.Embed method), 122
- set\_image() (novus.Embed method), 122
- set\_thumbnail() (novus.Embed method), 122
- SEXUAL\_CONTENT (novus.AutoModerationKeywordPresetType attribute), 153
- SHORT (novus.TextInputStyle attribute), 158
- SHORT\_DATE (novus.TimestampFormat attribute), 158
- SHORT\_DATETIME (novus.TimestampFormat attribute), 158
- SHORT\_TIME (novus.TimestampFormat attribute), 158
- size (novus.Attachment attribute), 7
- slug (novus.Application attribute), 5
- SLURS (novus.AutoModerationKeywordPresetType attribute), 153
- SPAM (novus.AutoModerationTriggerType attribute), 153
- speak (novus.Permissions attribute), 161
- splash (novus.Guild attribute), 36
- splash (novus.GuildPreview attribute), 61
- splash (novus.PartialGuild attribute), 83
- splash\_hash (novus.Guild attribute), 36
- splash\_hash (novus.GuildPreview attribute), 61
- splash\_hash (novus.PartialGuild attribute), 83
- spoiler (novus.File attribute), 124
- staff (novus.UserFlags attribute), 162
- stage\_instance (novus.api.HTTPConnection attribute), 167
- STAGE\_INSTANCE (novus.EventEntityType attribute), 154
- stage\_instance\_create (novus.AuditLogEntryType attribute), 163
- STAGE\_INSTANCE\_CREATE (novus.AuditLogEventType attribute), 152
- stage\_instance\_delete (novus.AuditLogEntryType attribute), 164
- STAGE\_INSTANCE\_DELETE (novus.AuditLogEventType attribute), 152
- stage\_instance\_update (novus.AuditLogEntryType attribute), 163
- STAGE\_INSTANCE\_UPDATE (novus.AuditLogEventType attribute), 152
- StageHTTPConnection (class in novus.api.stage\_instance), 173
- StageInstance (class in novus), 103
- STANDARD (novus.StickerType attribute), 158
- start() (novus.ext.client.Loop method), 189
- start\_behavior (novus.ext.client.Loop attribute), 188
- start\_thread\_from\_message() (novus.api.channel.ChannelHTTPConnection method), 170
- start\_thread\_in\_forum\_channel() (novus.api.channel.ChannelHTTPConnection method), 170
- start\_thread\_without\_message() (novus.api.channel.ChannelHTTPConnection method), 170
- start\_time (novus.ScheduledEvent attribute), 99
- state (novus.Activity attribute), 3
- state (novus.BaseGuild attribute), 13
- state (novus.ext.client.Client attribute), 178
- Status (class in novus), 158
- status (novus.ScheduledEvent attribute), 99
- status (novus.User attribute), 110
- Sticker (class in novus), 104
- sticker (novus.api.HTTPConnection attribute), 167
- sticker\_create (novus.AuditLogEntryType attribute), 164
- STICKER\_CREATE (novus.AuditLogEventType attribute), 152
- sticker\_delete (novus.AuditLogEntryType attribute), 164
- STICKER\_DELETE (novus.AuditLogEventType attribute), 152
- sticker\_items (novus.Message attribute), 78
- sticker\_update (novus.AuditLogEntryType attribute), 164

- 164  
 STICKER\_UPDATE (*novus.AuditLogEventType* attribute), 152  
 StickerFormat (*class in novus*), 157  
 StickerHTTPConnection (*class in novus.api.sticker*), 173  
 stickers (*novus.Guild* attribute), 40  
 stickers (*novus.GuildPreview* attribute), 62  
 StickerType (*class in novus*), 157  
 stop() (*novus.ext.client.Loop* method), 189  
 stream (*novus.Permissions* attribute), 161  
 STREAMING (*novus.ActivityType* attribute), 159  
 STRING (*novus.ApplicationOptionType* attribute), 151  
 STRING\_SELECT (*novus.ComponentType* attribute), 154  
 StringSelectMenu (*class in novus*), 127  
 style (*novus.Button* attribute), 127  
 style (*novus.TextInput* attribute), 134  
 SUB\_COMMAND (*novus.ApplicationOptionType* attribute), 151  
 SUB\_COMMAND\_GROUP (*novus.ApplicationOptionType* attribute), 151  
 SUCCESS (*novus.ButtonStyle* attribute), 153  
 suppress (*novus.VoiceState* attribute), 112  
 suppress\_guild\_reminder\_notifications (*novus.MessageFlags* attribute), 160  
 suppress\_guild\_reminder\_notifications (*novus.SystemChannelFlags* attribute), 160  
 suppress\_join\_notification\_replies (*novus.MessageFlags* attribute), 160  
 suppress\_join\_notification\_replies (*novus.SystemChannelFlags* attribute), 160  
 suppress\_join\_notifications (*novus.MessageFlags* attribute), 160  
 suppress\_join\_notifications (*novus.SystemChannelFlags* attribute), 160  
 suppress\_premium\_subscriptions (*novus.MessageFlags* attribute), 160  
 suppress\_premium\_subscriptions (*novus.SystemChannelFlags* attribute), 160  
 sync\_commands() (*novus.ext.client.Client* method), 181  
 system (*novus.GuildMember* attribute), 55  
 system (*novus.User* attribute), 109  
 system\_channel\_flags (*novus.Guild* attribute), 38  
 system\_channel\_id (*novus.Guild* attribute), 38  
 SystemChannelFlags (*class in novus*), 160
- ## T
- tags (*novus.Application* attribute), 5  
 tags (*novus.Role* attribute), 98  
 target (*novus.AuditLogEntry* attribute), 8  
 target (*novus.ContextComandData* attribute), 148  
 target\_id (*novus.AuditLogEntry* attribute), 8  
 task (*novus.ext.client.Loop* attribute), 189  
 Team (*class in novus*), 107  
 team (*novus.Application* attribute), 5  
 team\_id (*novus.TeamMember* attribute), 108  
 team\_pseudo\_user (*novus.UserFlags* attribute), 162  
 TeamMember (*class in novus*), 108  
 temporary (*novus.Invite* attribute), 75  
 terms\_of\_service\_url (*novus.Application* attribute), 4  
 TEXT\_INPUT (*novus.ComponentType* attribute), 154  
 TextInput (*class in novus*), 134  
 TextInputType (*class in novus*), 158  
 thread (*novus.Message* attribute), 78  
 thread\_create (*novus.AuditLogEntryType* attribute), 164  
 THREAD\_CREATE (*novus.AuditLogEventType* attribute), 152  
 THREAD\_CREATED (*novus.MessageType* attribute), 157  
 thread\_delete (*novus.AuditLogEntryType* attribute), 164  
 THREAD\_DELETE (*novus.AuditLogEventType* attribute), 152  
 thread\_id (*novus.ThreadMember* attribute), 108  
 THREAD\_STARTER\_MESSAGE (*novus.MessageType* attribute), 157  
 thread\_update (*novus.AuditLogEntryType* attribute), 164  
 THREAD\_UPDATE (*novus.AuditLogEventType* attribute), 152  
 ThreadMember (*class in novus*), 108  
 thumbnail (*novus.Embed* attribute), 121  
 TIER\_1 (*novus.PremiumTier* attribute), 157  
 TIER\_2 (*novus.PremiumTier* attribute), 157  
 TIER\_3 (*novus.PremiumTier* attribute), 157  
 TIMEOUT (*novus.AutoModerationActionType* attribute), 153  
 timeout\_until (*novus.GuildMember* attribute), 57  
 timestamp (*novus.Embed* attribute), 120  
 timestamp (*novus.Message* attribute), 76  
 TimestampFormat (*class in novus*), 158  
 title (*novus.Embed* attribute), 120  
 to\_application\_command\_option() (*novus.ext.client.Command* method), 184  
 to\_application\_command\_option() (*novus.ext.client.CommandGroup* method), 187  
 token (*novus.Interaction* attribute), 142  
 token (*novus.Webhook* attribute), 114  
 topic (*novus.StageInstance* attribute), 103  
 trigger\_metadata (*novus.AutoModerationRule* attribute), 10  
 trigger\_type (*novus.AutoModerationRule* attribute), 10  
 trigger\_typing() (*novus.Channel* method), 31  
 trigger\_typing\_indicator() (*novus.api.channel.ChannelHTTPConnection*

method), 169  
 try\_id() (in module novus.utils), 164  
 try\_object() (in module novus.utils), 165  
 try\_snowflake() (in module novus.utils), 164  
 tts (novus.Message attribute), 76  
 type (novus.Activity attribute), 3  
 type (novus.ApplicationCommand attribute), 137  
 type (novus.ApplicationCommandData attribute), 148  
 type (novus.ApplicationCommandOption attribute), 140  
 type (novus.AutoModerationAction attribute), 9  
 type (novus.Channel attribute), 26  
 type (novus.ContextCommandData attribute), 147  
 type (novus.Embed attribute), 120  
 type (novus.ext.client.Command attribute), 183  
 type (novus.Interaction attribute), 141  
 type (novus.Message attribute), 77  
 type (novus.MessageInteraction attribute), 147  
 type (novus.PartialApplicationCommand attribute), 136  
 type (novus.PermissionOverwrite attribute), 125  
 type (novus.Sticker attribute), 105  
 typing() (novus.Channel method), 31

## U

unban() (novus.BaseGuild method), 26  
 unban() (novus.Guild method), 54  
 unban() (novus.GuildPreview method), 74  
 unban() (novus.PartialGuild method), 96  
 unicode\_emoji (novus.Role attribute), 97  
 unpin() (novus.Message method), 81  
 unpin() (novus.WebhookMessage method), 119  
 unpin\_message() (novus.api.channel.ChannelHTTPConnection  
method), 170  
 update() (novus.Embed method), 122  
 update() (novus.GuildInteraction method), 146  
 update() (novus.Interaction method), 144  
 update\_application\_role\_records()  
 (novus.api.application\_role\_connection\_metadata  
method), 168  
 UPDATE\_MESSAGE (novus.InteractionResponseType  
attribute), 156  
 update\_user\_application\_role\_connection()  
 (novus.api.user.UserHTTPConnection  
method), 174  
 url (novus.Activity attribute), 3  
 url (novus.api.\_route.Route attribute), 165  
 url (novus.Attachment attribute), 7  
 url (novus.Button attribute), 127  
 URL (novus.ButtonStyle attribute), 153  
 url (novus.Embed attribute), 120  
 use\_application\_commands (novus.Permissions at-  
tribute), 161  
 use\_embedded\_activites (novus.Permissions at-  
tribute), 162

use\_external\_emojis (novus.Permissions attribute),  
161  
 use\_external\_stickers (novus.Permissions at-  
tribute), 162  
 use\_vad (novus.Permissions attribute), 161  
 User (class in novus), 109  
 user (novus.api.HTTPConnection attribute), 167  
 USER (novus.ApplicationCommandType attribute), 150  
 USER (novus.ApplicationOptionType attribute), 151  
 user (novus.AuditLogEntry attribute), 8  
 user (novus.GuildBan attribute), 54  
 user (novus.Interaction attribute), 142  
 user (novus.MessageInteraction attribute), 147  
 user (novus.TeamMember attribute), 108  
 user (novus.VoiceState attribute), 112  
 user\_count (novus.ScheduledEvent attribute), 100  
 user\_id (novus.AuditLogEntry attribute), 8  
 USER\_JOIN (novus.MessageType attribute), 157  
 USER\_SELECT (novus.ComponentType attribute), 154  
 UserFlags (class in novus), 162  
 UserHTTPConnection (class in novus.api.user), 173  
 username (novus.GuildMember attribute), 54  
 username (novus.User attribute), 109  
 UserPremiumType (class in novus), 158  
 users (novus.InteractionResolved attribute), 149  
 UserSelectMenu (class in novus), 131  
 uses (novus.Invite attribute), 74

## V

vanity\_url\_code (novus.PartialGuild attribute), 83  
 value (novus.ApplicationCommandChoice attribute),  
138  
 value (novus.SelectOption attribute), 133  
 value (novus.TextInput attribute), 135  
 values (novus.MessageComponentData attribute), 149  
 vanity\_url\_code (novus.Guild attribute), 39  
 ApplicationRoleGuildConnection (novus.Guild attribute), 37  
 verification\_level (novus.PartialGuild attribute), 83  
 verification\_pending\_guild\_limit  
 (novus.ApplicationFlags attribute), 160  
 VerificationLevel (class in novus), 158  
 verified (novus.GuildMember attribute), 56  
 verified (novus.User attribute), 110  
 verified\_bot (novus.UserFlags attribute), 162  
 verified\_developer (novus.UserFlags attribute), 162  
 verify\_key (novus.Application attribute), 5  
 version (novus.ApplicationCommand attribute), 138  
 VERY\_HIGH (novus.VerificationLevel attribute), 158  
 video (novus.Embed attribute), 121  
 VideoQualityMode (class in novus), 155  
 view\_audit\_log (novus.Permissions attribute), 161  
 view\_channel (novus.Permissions attribute), 161  
 view\_guild\_insights (novus.Permissions attribute),  
161

voice (*novus.api.HTTPConnection* attribute), 167  
VOICE (*novus.EventEntityType* attribute), 154  
voice (*novus.GuildMember* attribute), 57  
VOICE\_PING (*novus.GatewayOpcode* attribute), 155  
VOICE\_STATE (*novus.GatewayOpcode* attribute), 155  
VoiceHTTPConnection (*class in novus.api.voice*), 174  
VoiceState (*class in novus*), 112

## W

wait\_until\_ready (*novus.ext.client.Loop* attribute),  
189  
wait\_until\_ready() (*novus.ext.client.Client* method),  
179  
walk\_components() (*in module novus.utils*), 165  
WATCHING (*novus.ActivityType* attribute), 159  
Webhook (*class in novus*), 113  
webhook (*novus.api.HTTPConnection* attribute), 167  
webhook\_create (*novus.AuditLogEntryType* attribute),  
163  
WEBHOOK\_CREATE (*novus.AuditLogEventType* attribute),  
152  
webhook\_delete (*novus.AuditLogEntryType* attribute),  
163  
WEBHOOK\_DELETE (*novus.AuditLogEventType* attribute),  
152  
webhook\_id (*novus.Message* attribute), 77  
webhook\_update (*novus.AuditLogEntryType* attribute),  
163  
WEBHOOK\_UPDATE (*novus.AuditLogEventType* attribute),  
152  
WebhookHTTPConnection (*class in novus.api.webhook*),  
174  
WebhookMessage (*class in novus*), 116  
welcome\_screen (*novus.Guild* attribute), 40  
WelcomeScreen (*class in novus*), 119  
WelcomeScreenChannel (*class in novus*), 119  
widget\_channel\_id (*novus.Guild* attribute), 37  
widget\_enabled (*novus.Guild* attribute), 37