
discord-py-slash-command

eunwoo1104

May 29, 2021

CONTENTS:

1 Quickstart	3
2 Getting Started	5
2.1 Where do we start?	5
2.2 Making a slash command.	5
3 Migration	11
3.1 Migrate To V1.1.0	11
3.2 Migrate To 1.0.9	12
4 discord_slash package	15
4.1 Submodules	15
4.2 Module contents	33
5 discord_slash Events	35
6 discord_slash.utils package	37
6.1 Submodules	37
6.2 Module contents	41
7 Frequently Asked Questions	43
7.1 Why don't my slash commands show up?	43
7.2 How to add slash commands?	43
7.3 How to delete slash commands?	44
7.4 What is the difference between ctx.send and ctx.channel.send?	44
7.5 Can I use something of discord.py's in this extension?	45
7.6 Any more questions?	45
8 Indices and tables	47
Python Module Index	49
Index	51

Hello there! Welcome to the official documentation of our library extension made for discord.py: being able to use Discord Slash Commands.

Before we start going into the advanced stuff, it is highly recommended to check out the [quickstart](#) page first from here or below in the contents.

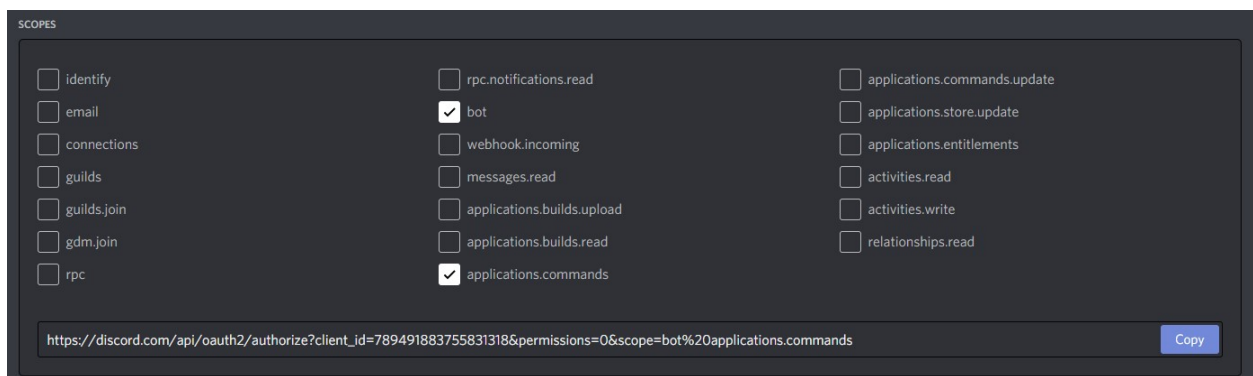
If there are any questions that you have about the documentation of this library extension that the docs do not currently cover over, please feel free to reach out to others on the [Discord](#)!

QUICKSTART

Before doing anything, it is highly recommended to read discord.py's quickstart. You can find it by clicking [this here](#). Firstly, we will begin by installing the python library extension for discord.py:

```
pip install -U discord-py-slash-command
```

Then, let's invite the bot. See discord.py's bot account create tutorial. After reading that, there is one more step before inviting your bot. The second step will now be setting your scope correctly for the bot to properly recognize slash commands, as shown here:



SCOPE

☐ identify	☐ rpc.notifications.read	☐ applications.commands.update
☐ email	☒ bot	☐ applications.store.update
☐ connections	☐ webhook.incoming	☐ applications.entitlements
☐ guilds	☐ messages.read	☐ activities.read
☐ guilds.join	☐ applications.builds.upload	☐ activities.write
☐ gdm.join	☐ applications.builds.read	☐ relationships.read
☐ rpc	☒ applications.commands	

https://discord.com/api/oauth2/authorize?client_id=789491883755831318&permissions=0&scope=bot%20applications.commands Copy

Then, invite your bot to your guild.

Now, let's create a simple bot. Create one Python code file. For this example, main.py will be used.

```
import discord
from discord_slash import SlashCommand # Importing the newly installed library.

client = discord.Client(intents=discord.Intents.all())
slash = SlashCommand(client, sync_commands=True) # Declares slash commands through the
↪ client.

@client.event
async def on_ready():
    print("Ready!")

client.run("your_bot_token_here")
```

Let's give this a run. When you run this code, you'll see... nothing but Ready!.

That's completely normal. Why is that? Well, it's because we haven't defined any actual slash commands just yet. We can do that by adding this code shown here:

```
"""
    Make sure this code is added before the client.run() call!
    It also needs to be under on_ready, otherwise, this will not work.
"""

guild_ids = [789032594456576001] # Put your server ID in this array.

@slash.slash(name="ping", guild_ids=guild_ids)
async def _ping(ctx): # Defines a new "context" (ctx) command called "ping."
    await ctx.send(f"Pong! ({client.latency*1000}ms)")
```

Note: In this example we responded directly to the interaction, however if you want to delay the response (if you need more than 3 seconds before sending a message) you can defer the response for up to 15 minutes with `ctx.defer()`, this displays a “Bot is thinking” message. However do not defer the response if you will be able to respond (send) within three seconds as this will cause a message to flash up

Let’s compare some of the major code differences between the prior examples in order to explain what’s going on here:

- `guild_ids = [789032594456576001]`: This is for adding your command as a guild command.

It is very important for us to make sure that we’re declaring this part of the `@slash.slash` decorator if we’re wanting to declare a guild command and not a **global** one. The reason as for why this is needed is because the Discord Bot API can take up to 1 hour to register a global command that is called via. the code here when this key-word argument is not passed.

- `@slash.slash(name="ping", ... ~ await ctx.send(...)`: This adds a new slash command.

This command basically sends a request to the API declaring a new command to exist as an HTTP request through the Bot v8 API.

Congratulations! You just created a very simple slash command bot! While, yes, this quickstart doesn’t cover everything, this still shows the basics of the library extension. In order to learn more about the advanced parts of adding slash commands, it is recommended to check out the other sections of our docs.

Still have any questions? Feel free to join the Discord server by clicking [this](#).

GETTING STARTED

2.1 Where do we start?

Before we begin getting started on everything else, it is recommended to check out the [quickstart](#) page first to get a basic grip on making slash commands for your bot.

2.2 Making a slash command.

2.2.1 The basics.

First, let's explain by how commands are parsed through the Discord Bot API.

As you may know, Discord relies a lot on the interaction of HTTP Requests and JSON tables. As is with the case here, commands are the exact same way with having JSON tables to structure the design of it for Discord to understand. We can apply this information likewise with how slash commands are to be designed in the Python code. Below attached is from the *Discord Developer Portal* on Slash Commands for showing how they are designed.

Field	Type	Description
name	string	1-32 character name matching <code>^[a-z-]{1,32}\$</code> .
de- scrip- tion	string	1-100 character description.
op- tions?	array of ApplicationCommandOption	if the option is a subcommand or subcommand group type, this nested options will be the parameters.

This table shows us the way that Discord handles the structure of commands for slash commands through their Bot API. For visualization purposes, we'll quickly make a JSON example (although we won't be using it) in order to explain how this works:

```
{
  "name": "test",
  "description": "This is just a test command, nothing more.",
}
```

Now that we have a basic understanding of how the JSON table works, we can take this knowledge and convert it into a decorator method for the Python code as shown below:

```
@slash.slash(name="test",
              description="This is just a test command, nothing more.")
async def test(ctx):
    await ctx.send(content="Hello World!")
```

Now that we've gone over how Discord handles the declaration of slash commands through their Bot API, let's go over what some of the other things mean within the *logical* part of our code, the command function:

2.2.2 Giving some options for variety.

The next thing that we will begin to talk about is the implementation of options, otherwise well-known as “arguments” in discord.py commands.

The JSON structure of options are designed up to be similar to the same premise of how a slash command is declared. Below is the given table of how an option JSON would appear as:

Field	Type	Description
type	int	value of <code>ApplicationCommandOptionType</code> .
name	string	1-32 character name matching <code>^[\\w-]{1,32}\$</code> .
de- scrip- tion	string	1-100 character description.
de- fault?	bool	the first required option for the user to complete—only one option can be default.
re- quired?	bool	if the parameter is required or optional—default <code>false</code> .
choices?	array of <code>ApplicationCommandOptionChoice</code>	choices for <code>string</code> and <code>int</code> types for the user to pick from.
op- tions?	array of <code>ApplicationCommandOption</code>	if the option is a subcommand or subcommand group type, this nested options will be the parameters.

Now we have an idea of how options are declared. With this in mind, let's quickly make a JSON example in order to visualize this concept even further:

```
{
  "name": "test",
  "description": "This is just a test command, nothing more.",
  "options": [
    {
      "name": "optone",
      "description": "This is the first option we have.",
      "type": 3,
      "required": "false"
    }
  ]
}
```

While the table in the basics mentions an array in particular called `ApplicationCommandOptionType`, there isn't that much of an explanation on how this works. Let's put this into better laymen terms on what this means with a table below showing all of these values:

Name	Value
SUB_COMMAND	1
SUB_COMMAND_GROUP	2
STRING	3
INTEGER	4
BOOLEAN	5
USER	6
CHANNEL	7
ROLE	8

The purpose of having the `ApplicationCommandOptionType` value passed into our option JSON structure is so that we can help the Discord UI understand what kind of value we're inputting here. For instance, if we're wanting to put in a string response, we'll pass the ID 3 so that the UI of Discord chat bar knows to format it visually this way. If we're looking for a user, then we'll pass ID 6 so that it presents us with a list of users in our server instead, making it easier on our lives.

This is not to be confused, however, with formatting the response type itself. This is merely a method so that the API wrapper can help us with passing the correct type or instance variable with the arguments of the command function's code.

Now, we can finally visualize this by coding an example of this being used in the Python code shown below.

```
from discord_slash.utils.manage_commands import create_option

@slash.slash(name="test",
             description="This is just a test command, nothing more.",
             options=[
                 create_option(
                     name="optone",
                     description="This is the first option we have.",
                     option_type=3,
                     required=False
                 )
             ])
async def test(ctx, optone: str):
    await ctx.send(content=f"I got you, you said {optone}!")
```

Additionally, we could also declare the type of our command's option through this method shown here:

```
from discord_slash.model import SlashCommandOptionType

(...)

option_type=SlashCommandOptionType.STRING
```

2.2.3 More in the option? Give them a choice.

Alas, there is also a way to give even more information to options with Discord's Slash Commands: a choice. Not like something that you're given at birth of when you become of legal age as an adult, we're not here to give you *that* kind of life advice, but the choice of what value you want your option to rather pass. Below is a table that shows the JSON structure of how choices are represented for an option:

Field	Type	Description
name	string	1-32 character choice name.
value	string or int	value of the choice, up to 100 characters if string.

This time, only 2 fields are able to be passed for this. Below is a JSON example of how this would be designed:

```
{
  "name": "ChoiceOne",
  "value": "Hello command, this is my value!"
}
```

To make it really simple, the `name` field is only to be used for how you want the choice to be presented through Discord's UI. It's the "appearance" of how you want your choice shown, not the actual returned value of it. Hence, this is why `value` is the second field passed for that, which can be either in the form of a string or integer. Below is an implementation of this design in the Python code:

```
from discord_slash.utils.manage_commands import create_option, create_choice

@slash.slash(name="test",
             description="This is just a test command, nothing more.",
             options=[
                 create_option(
                     name="optone",
                     description="This is the first option we have.",
                     option_type=3,
                     required=False,
                     choices=[
                         create_choice(
                             name="ChoiceOne",
                             value="DOGE!"
                         ),
                         create_choice(
                             name="ChoiceTwo",
                             value="NO DOGE"
                         )
                     ]
                 )
             ])
async def test(ctx, optone: str):
    await ctx.send(content=f"Wow, you actually chose {optone}?:(")
```

2.2.4 Want to restrict access? Setup permissions!

Slash commands also supports ability to set permissions to allow only certain roles and/or users to run a slash command. Permissions can be applied to both global and guild based commands. They are defined per guild ,per top-level command (the base command for subcommands), and each guild can have multiple permissions. Here table that shows the JSON structure of how permissions are represented:

Field	Type	Description
id	int	Snowflake value of type specified. Represents the target to apply permissions on.
type	int	An <code>ApplicationCommandPermissionType</code> .
permission	boolean	<code>True</code> to allow, <code>False</code> to disallow.

How the type parameter works is very simple. Discord has many ids to represent different things. As you can set permissions to apply for User or Role, `ApplicationCommandPermissionType` is used. It's a number and following table shows the supported id types for permissions:

Name	Value
Role	1
User	2

This is an example of how a single permission will look when represented as a json object:

```
{
  "id": 12345678,
  "type": 1,
  "permission": True
}
```

Now, let take a look at an example. The slash command decorator have a permissions parameter where it takes in a dictionary. The key being the guild id to apply permissions on, and value being the list of permissions to apply. For each permission, we can use the handy `create_permission` method to build the permission json explain above.

In this case, we are setting 2 permissions for guild with id of 12345678. Firstly, we are allowing role with id 99999999 and disallowing user with id 88888888 from running the slash command.

```
from discord_slash.utils.manage_commands import create_permission
from discord_slash.model import SlashCommandPermissionType

@slash.slash(name="test",
             description="This is just a test command, nothing more.",
             permissions={
                 12345678: [
                     create_permission(99999999, SlashCommandPermissionType.ROLE, True),
                     create_permission(88888888, SlashCommandPermissionType.USER, False)
                 ]
             })
async def test(ctx):
    await ctx.send(content="Hello World!")
```

Alternatively, you can use the `@slash.permission` decorator to define your guild permissions for the command as show in the following example:

```
from discord_slash.utils.manage_commands import create_permission
from discord_slash.model import SlashCommandPermissionType

@slash.slash(name="test",
             description="This is just a test command, nothing more.")
@slash.permission(guild_id=12345678,
                 permissions=[
                     create_permission(99999999, SlashCommandPermissionType.ROLE, True),
                     create_permission(88888888, SlashCommandPermissionType.USER, False)
                 ])
async def test(ctx):
    await ctx.send(content="Hello World!")
```

MIGRATION

This page contains instructions on how to migrate between versions with breaking changes.

3.1 Migrate To V1.1.0

Changes that you'll need to make in this version are mainly because of a new ui from discord, more info [here](#)

3.1.1 Responding / Deferring

In regards to `SlashContext` the methods `.respond` and `.ack` have been removed, and `.defer` added. Deferring a message will allow you to respond for up to 15 mins, but you'll have to defer within three seconds. When you defer the user will see a "this bot is thinking" message until you send a message, This message can be ephemeral (hidden) or visible. `.defer` has a `hidden` parameter, which will make the deferred message ephemeral.

We suggest deferring if you might take more than three seconds to respond, but if you will call `.send` before three seconds then don't.

Note: You **must** respond with `.send` within 15 minutes of the command being invoked to avoid an "interaction failed" message, if you defer. This is especially relevant for bots that had 'invisible' commands, where the invocation was not shown. If you wish to have the invocation of the command not visible, send an ephemeral success message, and then do what you used to. It is no longer possible to "eat" the invocation. `ctx.channel.send` does **not** count as responding.

Example

```
# Before
@slash.slash(...)
async def command(ctx, ...):
    await ctx.respond()
    await ctx.send(...)

# After 1
@slash.slash(...)
async def command(ctx, ...):
    await ctx.send(...)

# After 2
@slash.slash(...)
```

(continues on next page)

(continued from previous page)

```
async def command(ctx, ...):
    await ctx.defer()
    # Process that takes time
    await ctx.send(...)
```

3.1.2 Sending hidden messages

The method `.send_hidden` on `SlashContext` has been removed. Use `.send(hidden = True, ...)` instead.

3.1.3 SlashContext

`ctx.sent` has been renamed to `ctx.responded`

3.2 Migrate To 1.0.9

1.0.9 Update includes many feature add/rewrite for easier and better usage. However, due to that, it includes many critical breaking changes.

This page will show how to deal with those changes.

3.2.1 SlashContext

`.send()` / `.delete()` / `.edit()`

Before:

```
# Case 1
await ctx.send(4, content="Hello, World! This is initial message.")
await ctx.edit(content="Or nevermind.")
await ctx.delete()
await ctx.send(content="This is followup message.")

# Case 2
await ctx.send(content="This is secret message.", complete_hidden=True)
```

After:

```
# Case 1
await ctx.respond() # This is optional, but still recommended to.
msg = await ctx.send("Hello, World! This is initial message.")
await msg.edit(content="Or nevermind.")
await msg.delete()
await ctx.send("This is followup message.")

# Case 2
await ctx.respond(eat=True) # Again, this is optional, but still recommended to.
await ctx.send("This is secret message.", hidden=True)
```

Objects of the command invoke

Before:

```
author_id = ctx.author.id if isinstance(ctx.author, discord.Member) else ctx.author
channel_id = ctx.channel.id if isinstance(ctx.channel, discord.TextChannel) else ctx.
↳channel
guild_id = ctx.guild.id if isinstance(ctx.guild, discord.Guild) else ctx.guild
...
```

After:

```
author_id = ctx.author_id
channel_id = ctx.channel_id
guild_id = ctx.guild_id
...
```

3.2.2 Auto-registering

We've changed the method of automatically registering commands to API to reduce the request amount and prevent rate limit. So, `auto_register` and `auto_delete` parameter is now removed. Please change your SlashContext params like this.

Before:

```
slash = SlashContext(..., auto_register=True, auto_delete=True) # Either one can be_
↳false.
```

After:

```
slash = SlashContext(..., sync_commands=True)
```

3.2.3 Cog Support

Before:

```
class Slash(commands.Cog):
    def __init__(self, bot):
        if not hasattr(bot, "slash"):
            # Creates new SlashCommand instance to bot if bot doesn't have.
            bot.slash = SlashCommand(bot, override_type=True)
            # Note that hasattr block is optional, meaning you might not have it.
            # Its completely fine, and ignore it.
            self.bot = bot
            self.bot.slash.get_cog_commands(self)

    def cog_unload(self):
        self.bot.slash.remove_cog_commands(self)

    ...
```

After:

```
class Slash(commands.Cog):
    def __init__(self, bot):
        self.bot = bot

    ...
```

As you can see *if not hasattr(...)*: block is removed, moving to main file like this is necessary.

```
bot = commands.Bot(...)
slash = SlashCommand(bot)
# No worries for not doing `bot.slash` because its automatically added now.
...

```

Auto-convert

It got deleted, so please remove all of it if you used it.

Also, we've added *connector* parameter, which is for helping passing options as kwargs if your command option is other than english.

Usage:

```
{
    "example-arg": "example_arg",
    "": "hour"
}
```

DISCORD_SLASH PACKAGE

4.1 Submodules

4.1.1 discord_slash.client module

```
class discord_slash.client.SlashCommand(client: Union[discord.client.Client,  
discord.ext.commands.bot.Bot], sync_commands: bool = False,  
delete_from_unused_guilds: bool = False, sync_on_cog_reload:  
bool = False, override_type: bool = False, application_id:  
Optional[int] = None)
```

Bases: `object`

Slash command handler class.

Parameters

- **client** (`Union[discord.Client, discord.ext.commands.Bot]`) – discord.py Client or Bot instance.
- **sync_commands** (`bool`) – Whether to sync commands automatically. Default *False*.
- **delete_from_unused_guilds** (`bool`) – If the bot should make a request to set no commands for guilds that haven't got any commands registered in :class:SlashCommand. Default *False*.
- **sync_on_cog_reload** (`bool`) – Whether to sync commands on cog reload. Default *False*.
- **override_type** (`bool`) – Whether to override checking type of the client and try register event.
- **application_id** (`int`) – The application id of the bot, required only when the application id and bot id are different. (old bots)

Note: If `sync_on_cog_reload` is enabled, command syncing will be triggered when `discord.ext.commands.Bot.reload_extension()` is triggered.

Variables

- **_discord** – Discord client of this client.
- **commands** – Dictionary of the registered commands via `slash()` decorator.
- **req** – `http.SlashCommandRequest` of this client.
- **logger** – Logger of this client.

- **sync_commands** – Whether to sync commands automatically.
- **sync_on_cog_reload** – Whether to sync commands on cog reload.
- **has_listener** – Whether discord client has listener add function.

get_cog_commands(*cog: discord.ext.commands.Cog*)
Gets slash command from `discord.ext.commands.Cog`.

Note: Since version 1.0.9, this gets called automatically during cog initialization.

Parameters cog (*discord.ext.commands.Cog*) – Cog that has slash commands.

remove_cog_commands(*cog*)
Removes slash command from `discord.ext.commands.Cog`.

Note: Since version 1.0.9, this gets called automatically during cog de-initialization.

Parameters cog (*discord.ext.commands.Cog*) – Cog that has slash commands.

async to_dict()
Converts all commands currently registered to *SlashCommand* to a dictionary. Returns a dictionary in the format:

```
{
    "global" : [], # list of global commands
    "guild" : {
        0000: [] # list of commands in the guild 0000
    }
}
```

Commands are in the format specified by discord [here](#)

async sync_all_commands(*delete_from_unused_guilds=False, delete_perms_from_unused_guilds=False*)
Matches commands registered on Discord to commands registered here. Deletes any commands on Discord but not here, and registers any not on Discord. This is done with a *put* request. A PUT request will only be made if there are changes detected. If `sync_commands` is `True`, then this will be automatically called.

Parameters

- **delete_from_unused_guilds** – If the bot should make a request to set no commands for guilds that haven't got any commands registered in `:class:SlashCommand`
- **delete_perms_from_unused_guilds** – If the bot should make a request to clear permissions for guilds that haven't got any permissions registered in `:class:SlashCommand`

add_slash_command(*cmd, name: Optional[str] = None, description: Optional[str] = None, guild_ids: Optional[List[int]] = None, options: Optional[list] = None, default_permission: bool = True, permissions: Optional[Dict[int, list]] = None, connector: Optional[dict] = None, has_subcommands: bool = False*)

Registers slash command to `SlashCommand`.

Warning: Just using this won't register slash command to Discord API. To register it, check `utils.manage_commands.add_slash_command()` or simply enable `sync_commands`.

Parameters

- **cmd** (*Coroutine*) – Command Coroutine.
- **name** (*str*) – Name of the slash command. Default name of the coroutine.
- **description** (*str*) – Description of the slash command. Defaults to command docstring or None.
- **guild_ids** (*List[int]*) – List of Guild ID of where the command will be used. Default None, which will be global command.
- **options** (*list*) – Options of the slash command. This will affect `auto_convert` and command data at Discord API. Default None.
- **default_permission** (*bool*) – Sets if users have permission to run slash command by default, when no permissions are set. Default True.
- **permissions** (*dict*) – Dictionary of permissions of the slash command. Key being target guild_id and value being a list of permissions to apply. Default None.
- **connector** (*dict*) – Kwargs connector for the command. Default None.
- **has_subcommands** (*bool*) – Whether it has subcommand. Default False.

add_subcommand(*cmd*, *base*, *subcommand_group*=None, *name*=None, *description*: *Optional[str]* = None, *base_description*: *Optional[str]* = None, *base_default_permission*: *bool* = True, *base_permissions*: *Optional[Dict[int, list]]* = None, *subcommand_group_description*: *Optional[str]* = None, *guild_ids*: *Optional[List[int]]* = None, *options*: *Optional[list]* = None, *connector*: *Optional[dict]* = None)

Registers subcommand to SlashCommand.

Parameters

- **cmd** (*Coroutine*) – Subcommand Coroutine.
- **base** (*str*) – Name of the base command.
- **subcommand_group** (*str*) – Name of the subcommand group, if any. Default None which represents there is no sub group.
- **name** (*str*) – Name of the subcommand. Default name of the coroutine.
- **description** (*str*) – Description of the subcommand. Defaults to command docstring or None.
- **base_description** (*str*) – Description of the base command. Default None.
- **default_permission** (*bool*) – Sets if users have permission to run base command by default, when no permissions are set. Default True.
- **base_permissions** (*dict*) – Dictionary of permissions of the slash command. Key being target guild_id and value being a list of permissions to apply. Default None.
- **subcommand_group_description** (*str*) – Description of the subcommand_group. Default None.
- **guild_ids** (*List[int]*) – List of guild ID of where the command will be used. Default None, which will be global command.

- **options** (*list*) – Options of the subcommand. This will affect `auto_convert` and command data at Discord API. Default `None`.
- **connector** (*dict*) – Kwargs connector for the command. Default `None`.

slash(***, *name*: *Optional[str]* = *None*, *description*: *Optional[str]* = *None*, *guild_ids*: *Optional[List[int]]* = *None*, *options*: *Optional[List[dict]]* = *None*, *default_permission*: *bool* = *True*, *permissions*: *Optional[dict]* = *None*, *connector*: *Optional[dict]* = *None*)

Decorator that registers coroutine as a slash command.

All decorator args must be passed as keyword-only args.

1 arg for command coroutine is required for `ctx(model.SlashContext)`, and if your slash command has some args, then those args are also required.

All args must be passed as keyword-args.

Note: If you don't pass *options* but has extra args, then it will automatically generate options. However, it is not recommended to use it since descriptions will be "No Description." or the command's description.

Warning: Unlike discord.py's command, **args*, keyword-only args, converters, etc. are not supported or behave differently.

Example:

```
@slash.slash(name="ping")
async def _slash(ctx): # Normal usage.
    await ctx.send(content=f"Pong! ({round(bot.latency*1000)}ms)")

@slash.slash(name="pick")
async def _pick(ctx, choice1, choice2): # Command with 1 or more args.
    await ctx.send(content=str(random.choice([choice1, choice2])))
```

To format the connector, follow this example.

```
{
    "example-arg": "example_arg",
    "": "hour"
    # Formatting connector is required for
    # using other than english for option parameter name
    # for in case.
}
```

Set discord UI's parameter name as key, and set command coroutine's arg name as value.

Parameters

- **name** (*str*) – Name of the slash command. Default name of the coroutine.
- **description** (*str*) – Description of the slash command. Default `None`.
- **guild_ids** (*List[int]*) – List of Guild ID of where the command will be used. Default `None`, which will be global command.
- **options** (*List[dict]*) – Options of the slash command. This will affect `auto_convert` and command data at Discord API. Default `None`.

- **default_permission** (*bool*) – Sets if users have permission to run slash command by default, when no permissions are set. Default `True`.
- **permissions** (*dict*) – Permission requirements of the slash command. Default `None`.
- **connector** (*dict*) – Kwargs connector for the command. Default `None`.

subcommand(***, *base*, *subcommand_group*=`None`, *name*=`None`, *description*: *Optional[str]* = `None`, *base_description*: *Optional[str]* = `None`, *base_desc*: *Optional[str]* = `None`, *base_default_permission*: *bool* = `True`, *base_permissions*: *Optional[dict]* = `None`, *subcommand_group_description*: *Optional[str]* = `None`, *sub_group_desc*: *Optional[str]* = `None`, *guild_ids*: *Optional[List[int]]* = `None`, *options*: *Optional[List[dict]]* = `None`, *connector*: *Optional[dict]* = `None`)

Decorator that registers subcommand.

Unlike `discord.py`, you don't need base command.

All args must be passed as keyword-args.

Note: If you don't pass *options* but has extra args, then it will automatically generate options. However, it is not recommended to use it since descriptions will be "No Description." or the command's description.

Warning: Unlike `discord.py`'s `command`, **args*, keyword-only args, converters, etc. are not supported or behave differently.

Example:

```
# /group say <str>
@slash.subcommand(base="group", name="say")
async def _group_say(ctx, _str):
    await ctx.send(content=_str)

# /group kick user <user>
@slash.subcommand(base="group",
                  subcommand_group="kick",
                  name="user")
async def _group_kick_user(ctx, user):
    ...
```

Parameters

- **base** (*str*) – Name of the base command.
- **subcommand_group** (*str*) – Name of the subcommand group, if any. Default `None` which represents there is no sub group.
- **name** (*str*) – Name of the subcommand. Default name of the coroutine.
- **description** (*str*) – Description of the subcommand. Default `None`.
- **base_description** (*str*) – Description of the base command. Default `None`.
- **base_desc** – Alias of `base_description`.
- **default_permission** (*bool*) – Sets if users have permission to run slash command by default, when no permissions are set. Default `True`.

- **permissions** (*dict*) – Permission requirements of the slash command. Default `None`.
- **subcommand_group_description** (*str*) – Description of the subcommand_group. Default `None`.
- **sub_group_desc** – Alias of `subcommand_group_description`.
- **guild_ids** (*List[int]*) – List of guild ID of where the command will be used. Default `None`, which will be global command.
- **options** (*List[dict]*) – Options of the subcommand. This will affect `auto_convert` and command data at Discord API. Default `None`.
- **connector** (*dict*) – Kwargs connector for the command. Default `None`.

permission(*guild_id: int, permissions: list*)

Decorator that add permissions. This will set the permissions for a single guild, you can use it more than once for each command. :param guild_id: ID of the guild for the permissions. :type guild_id: int :param permissions: Permission requirements of the slash command. Default `None`. :type permissions: dict

async process_options(*guild: discord.guild.Guild, options: list, connector: dict, temporary_auto_convert: Optional[dict] = None*) → *dict*

Processes Role, User, and Channel option types to discord.py's models.

Parameters

- **guild** (*discord.Guild*) – Guild of the command message.
- **options** (*list*) – Dict of options.
- **connector** – Kwargs connector.
- **temporary_auto_convert** – Temporary parameter, use this if options doesn't have type keyword.

Returns Union[list, dict]

async invoke_command(*func, ctx, args*)

Invokes command.

Parameters

- **func** – Command coroutine.
- **ctx** – Context.
- **args** – Args. Can be list or dict.

async on_socket_response(*msg*)

This event listener is automatically registered at initialization of this class.

Warning: DO NOT MANUALLY REGISTER, OVERRIDE, OR WHATEVER ACTION TO THIS COROUTINE UNLESS YOU KNOW WHAT YOU ARE DOING.

Parameters *msg* – Gateway message.

async handle_subcommand(*ctx: discord_slash.context.SlashContext, data: dict*)

Coroutine for handling subcommand.

Warning: Do not manually call this.

Parameters

- **ctx** – `model.SlashContext` instance.
- **data** – Gateway message.

async on_slash_command_error(ctx, ex)

Handles Exception occurred from invoking command.

Example of adding event:

```
@client.event
async def on_slash_command_error(ctx, ex):
    ...
```

Example of adding listener:

```
@bot.listen()
async def on_slash_command_error(ctx, ex):
    ...
```

Parameters

- **ctx** (`model.SlashContext`) – Context of the command.
- **ex** (*Exception*) – Exception from the command invoke.

Returns

4.1.2 discord_slash.cog_ext module

`discord_slash.cog_ext.cog_slash`(*, name: *Optional[str]* = None, description: *Optional[str]* = None, guild_ids: *Optional[List[int]]* = None, options: *Optional[List[dict]]* = None, default_permission: *bool* = True, permissions: *Optional[Dict[int, list]]* = None, connector: *Optional[dict]* = None)

Decorator for Cog to add slash command.

Almost same as `client.SlashCommand.slash()`.

Example:

```
class ExampleCog(commands.Cog):
    def __init__(self, bot):
        self.bot = bot

    @cog_ext.cog_slash(name="ping")
    async def ping(self, ctx: SlashContext):
        await ctx.send(content="Pong!")
```

Parameters

- **name** (*str*) – Name of the slash command. Default name of the coroutine.
- **description** (*str*) – Description of the slash command. Default None.

- **guild_ids** (*List[int]*) – List of Guild ID of where the command will be used. Default `None`, which will be global command.
- **options** (*List[dict]*) – Options of the slash command. This will affect `auto_convert` and command data at Discord API. Default `None`.
- **default_permission** (*bool*) – Sets if users have permission to run slash command by default, when no permissions are set. Default `True`.
- **permissions** (*dict*) – Dictionary of permissions of the slash command. Key being target `guild_id` and value being a list of permissions to apply. Default `None`.
- **connector** (*dict*) – Kwargs connector for the command. Default `None`.

`discord_slash.cog_ext.cog_subcommand(*, base, subcommand_group=None, name=None, description: Optional[str] = None, base_description: Optional[str] = None, base_desc: Optional[str] = None, base_default_permission: bool = True, base_permissions: Optional[Dict[int, list]] = None, subcommand_group_description: Optional[str] = None, sub_group_desc: Optional[str] = None, guild_ids: Optional[List[int]] = None, options: Optional[List[dict]] = None, connector: Optional[dict] = None)`

Decorator for Cog to add subcommand.

Almost same as `client.SlashCommand.subcommand()`.

Example:

```
class ExampleCog(commands.Cog):
    def __init__(self, bot):
        self.bot = bot

    @cog_ext.cog_subcommand(base="group", name="say")
    async def group_say(self, ctx: SlashContext, text: str):
        await ctx.send(content=text)
```

Parameters

- **base** (*str*) – Name of the base command.
- **subcommand_group** (*str*) – Name of the subcommand group, if any. Default `None` which represents there is no sub group.
- **name** (*str*) – Name of the subcommand. Default name of the coroutine.
- **description** (*str*) – Description of the subcommand. Default `None`.
- **base_description** (*str*) – Description of the base command. Default `None`.
- **base_desc** – Alias of `base_description`.
- **base_default_permission** (*bool*) – Sets if users have permission to run slash command by default, when no permissions are set. Default `True`.
- **base_permissions** (*dict*) – Dictionary of permissions of the slash command. Key being target `guild_id` and value being a list of permissions to apply. Default `None`.
- **subcommand_group_description** (*str*) – Description of the subcommand_group. Default `None`.
- **sub_group_desc** – Alias of `subcommand_group_description`.

- **guild_ids** (*List[int]*) – List of guild ID of where the command will be used. Default None, which will be global command.
- **options** (*List[dict]*) – Options of the subcommand. This will affect `auto_convert` and command data at Discord API. Default None.
- **connector** (*dict*) – Kwargs connector for the command. Default None.

4.1.3 discord_slash.context module

class discord_slash.context.SlashContext(*_http: discord_slash.http.SlashCommandRequest, _json: dict, _discord: Union[discord.client.Client, discord.ext.commands.bot.Bot], logger*)

Bases: `object`

Context of the slash command.

Kinda similar with `discord.ext.commands.Context`.

Warning: Do not manually init this model.

Variables

- **message** – Message that invoked the slash command.
- **name** – Name of the command.
- **args** – List of processed arguments invoked with the command.
- **kwargs** – Dictionary of processed arguments invoked with the command.
- **subcommand_name** – Subcommand of the command.
- **subcommand_group** – Subcommand group of the command.
- **interaction_id** – Interaction ID of the command message.
- **command_id** – ID of the command.
- **bot** – discord.py client.
- **_http** – *http.SlashCommandRequest* of the client.
- **_logger** – Logger instance.
- **deferred** – Whether the command is current deferred (loading state)
- **_deferred_hidden** – Internal var to check that state stays the same
- **responded** – Whether you have responded with a message to the interaction.
- **guild_id** – Guild ID of the command message. If the command was invoked in DM, then it is None
- **author_id** – User ID representing author of the command message.
- **channel_id** – Channel ID representing channel of the command message.
- **author** – User or Member instance of the command invoke.

property deferred

property guild: `Optional[discord.guild.Guild]`

Guild instance of the command invoke. If the command was invoked in DM, then it is None

Returns `Optional[discord.Guild]`

property channel: `Optional[Union[discord.channel.TextChannel, discord.channel.DMChannel]]`

Channel instance of the command invoke.

Returns `Optional[Union[discord.abc.GuildChannel, discord.abc.PrivateChannel]]`

async defer(*hidden: bool = False*)

‘Defers’ the response, showing a loading state to the user

Parameters **hidden** – Whether the deferred response should be ephemeral . Default False.

async send(*content: str = "", *, embed: Optional[discord.embeds.Embed] = None, embeds: Optional[List[discord.embeds.Embed]] = None, tts: bool = False, file: Optional[discord.file.File] = None, files: Optional[List[discord.file.File]] = None, allowed_mentions: Optional[discord.mentions.AllowedMentions] = None, hidden: bool = False, delete_after: Optional[float] = None*) → *discord_slash.model.SlashMessage*

Sends response of the slash command.

Warning:

- Since Release 1.0.9, this is completely changed. If you are migrating from older version, please make sure to fix the usage.
- You can’t use both `embed` and `embeds` at the same time, also applies to `file` and `files`.
- If you send files in the initial response, this will defer if it’s not been deferred, and then PATCH with the message

Parameters

- **content** (*str*) – Content of the response.
- **embed** (*discord.Embed*) – Embed of the response.
- **embeds** (*List[discord.Embed]*) – Embeds of the response. Maximum 10.
- **tts** (*bool*) – Whether to speak message using tts. Default False.
- **file** (*discord.File*) – File to send.
- **files** (*List[discord.File]*) – Files to send.
- **allowed_mentions** (*discord.AllowedMentions*) – AllowedMentions of the message.
- **hidden** (*bool*) – Whether the message is hidden, which means message content will only be seen to the author.
- **delete_after** (*float*) – If provided, the number of seconds to wait in the background before deleting the message we just sent. If the deletion fails, then it is silently ignored.

Returns `Union[discord.Message, dict]`

4.1.4 discord_slash.error module

exception discord_slash.error.SlashCommandError

Bases: [Exception](#)

All exceptions of this extension can be captured with this.

Note: discord.py doesn't trigger *on_command_error* event. Use this extension's *on_slash_command_error*.

exception discord_slash.error.RequestFailure(*status: int, msg: str*)

Bases: [discord_slash.error.SlashCommandError](#)

Request to Discord API has failed.

Note: Since release 1.0.8, this is only used at *utils.manage_commands*. *http.SlashCommandRequest* uses exception from discord.py such as [discord.HTTPException](#).

Variables

- **status** – Status code of failed response.
- **msg** – Message of failed response.

exception discord_slash.error.IncorrectFormat

Bases: [discord_slash.error.SlashCommandError](#)

Some formats are incorrect. See Discord API DOCS for proper format.

exception discord_slash.error.DuplicateCommand(*name: str*)

Bases: [discord_slash.error.SlashCommandError](#)

There is a duplicate command name.

exception discord_slash.error.DuplicateSlashClient

Bases: [discord_slash.error.SlashCommandError](#)

There are duplicate *SlashCommand* instances.

exception discord_slash.error.CheckFailure

Bases: [discord_slash.error.SlashCommandError](#)

Command check has failed.

exception discord_slash.error.IncorrectType

Bases: [discord_slash.error.SlashCommandError](#)

Type passed was incorrect

exception discord_slash.error.IncorrectCommandData

Bases: [discord_slash.error.SlashCommandError](#)

Incorrect data was passed to a slash command data object

exception discord_slash.error.AlreadyResponded

Bases: [discord_slash.error.SlashCommandError](#)

The interaction was already responded to

4.1.5 discord_slash.http module

class discord_slash.http.CustomRoute(*method, path, **parameters*)

Bases: discord.http.Route

discord.py's Route but changed BASE to use at slash command.

BASE = 'https://discord.com/api/v8'

class discord_slash.http.SlashCommandRequest(*logger, _discord, application_id*)

Bases: object

property application_id

put_slash_commands(*slash_commands: list, guild_id*)

Sends a slash command put request to the Discord API

slash_commands must contain all the commands ;param slash_commands: List of all the slash commands to make a put request to discord with. ;param guild_id: ID of the guild to set the commands on. Pass *None* for the global scope.

remove_slash_command(*guild_id, cmd_id*)

Sends a slash command delete request to Discord API.

Parameters

- **guild_id** – ID of the guild to add command. Pass *None* to add global command.
- **cmd_id** – ID of the command.

Returns Response code of the request.

get_all_commands(*guild_id=None*)

Sends a slash command get request to Discord API for all commands.

Parameters **guild_id** – ID of the guild to add command. Pass *None* to add global command.

Returns JSON Response of the request.

get_all_guild_commands_permissions(*guild_id*)

Sends a slash command get request to Discord API for all permissions of a guild.

Parameters **guild_id** – ID of the target guild to get registered command permissions of.

Returns JSON Response of the request.

update_guild_commands_permissions(*guild_id, perms_dict*)

Sends a slash command put request to the Discord API for setting all command permissions of a guild.

Parameters **guild_id** – ID of the target guild to register command permissions.

Returns JSON Response of the request.

add_slash_command(*guild_id, cmd_name: str, description: str, options: Optional[list] = None*)

Sends a slash command add request to Discord API.

Parameters

- **guild_id** – ID of the guild to add command. Pass *None* to add global command.
- **cmd_name** – Name of the command. Must be 3 or longer and 32 or shorter.
- **description** – Description of the command.
- **options** – List of the function.

Returns JSON Response of the request.

command_request(*method, guild_id, url_ending="", **kwargs*)

Sends a command request to discord (post, get, delete, etc)

Parameters

- **method** – HTTP method.
- **guild_id** – ID of the guild to make the request on. *None* to make a request on the global scope.
- **url_ending** – String to append onto the end of the url.
- ****kwargs** – Kwargs to pass into discord.py’s [request function](#)

post_followup(*_resp, token, files: Optional[List[discord.file.File]] = None*)

Sends command followup response POST request to Discord API.

Parameters

- **_resp** (*dict*) – Command response.
- **token** – Command message token.
- **files** (*List[discord.File]*) – Files to send. Default *None*

Returns Coroutine

post_initial_response(*_resp, interaction_id, token*)

Sends an initial “POST” response to the Discord API.

Parameters

- **_resp** (*dict*) – Command response.
- **interaction_id** – Interaction ID.
- **token** – Command message token.

Returns Coroutine

command_response(*token, use_webhook, method, interaction_id=None, url_ending="", **kwargs*)

Sends a command response to discord (POST, PATCH, DELETE)

Parameters

- **token** – Interaction token
- **use_webhook** – Whether to use webhooks
- **method** – The HTTP request to use
- **interaction_id** – The id of the interaction
- **url_ending** – String to append onto the end of the url.
- ****kwargs** – Kwargs to pass into discord.py’s [request function](#)

Returns Coroutine

request_with_files(*_resp, files: List[discord.file.File], token, method, url_ending=""*)

edit(*_resp, token, message_id='@original', files: Optional[List[discord.file.File]] = None*)

Sends edit command response PATCH request to Discord API.

Parameters

- **_resp** (*dict*) – Edited response.
- **token** – Command message token.

- **message_id** – Message ID to edit. Default initial message.
- **files** (*List*[*discord.File*]) – Files. Default None

Returns Coroutine

delete(*token, message_id='@original'*)

Sends delete command response POST request to Discord API.

Parameters

- **token** – Command message token.
- **message_id** – Message ID to delete. Default initial message.

Returns Coroutine

4.1.6 discord_slash.model module

class discord_slash.model.**ChoiceData**(*name, value*)

Bases: *object*

Command choice data object

Variables

- **name** – Name of the choice, this is what the user will see
- **value** – Values of the choice, this is what discord will return to you

class discord_slash.model.**OptionData**(*name, description, required=False, choices=None, options=None, **kwargs*)

Bases: *object*

Command option data object

Variables

- **name** – Name of the option.
- **description** – Description of the option.
- **required** – If the option is required.
- **choices** – A list of *ChoiceData*, cannot be present on subcommand groups
- **options** – List of *OptionData*, this will be present if it's a subcommand group

class discord_slash.model.**CommandData**(*name, description, options=None, id=None, application_id=None, version=None, **kwargs*)

Bases: *object*

Slash command data object

Variables

- **name** – Name of the command.
- **description** – Description of the command.
- **options** – List of *OptionData*.
- **id** – Command id, this is received from discord so may not be present
- **application_id** – The application id of the bot, required only when the application id and bot id are different. (old bots)

class discord_slash.model.CommandObject(name, cmd)

Bases: `object`

Slash command object of this extension.

Warning: Do not manually init this model.

Variables

- **name** – Name of the command.
- **func** – The coroutine of the command.
- **description** – Description of the command.
- **allowed_guild_ids** – List of the allowed guild id.
- **options** – List of the option of the command. Used for *auto_register*.
- **connector** – Kwargs connector of the command.
- **__commands_checks__** – Check of the command.

async invoke(*args)

Invokes the command.

Parameters args – Args for the command.

Raises .error.CheckFailure

add_check(func)

Adds check to the command.

Parameters func – Any callable. Coroutines are supported.

remove_check(func)

Removes check to the command.

Note: If the function is not found at the command check, it will ignore.

Parameters func – Any callable. Coroutines are supported.

async can_run(ctx) → bool

Whether the command can be run.

Parameters ctx (`context.SlashContext`) – SlashContext for the check running.

Returns bool

class discord_slash.model.BaseCommandObject(name, cmd)

Bases: `discord_slash.model.CommandObject`

BaseCommand object of this extension.

Note: This model inherits `model.CommandObject`, so this has every variables from that.

Warning: Do not manually init this model.

Variables

- **has_subcommands** – Indicates whether this base command has subcommands.
- **default_permission** – Indicates whether users should have permissions to run this command by default.
- **permissions** – Permissions to restrict use of this command.

add_check(*func*)

Adds check to the command.

Parameters **func** – Any callable. Coroutines are supported.

async can_run(*ctx*) → bool

Whether the command can be run.

Parameters **ctx** ([context.SlashContext](#)) – SlashContext for the check running.

Returns bool

async invoke(**args*)

Invokes the command.

Parameters **args** – Args for the command.

Raises `.error.CheckFailure`

remove_check(*func*)

Removes check to the command.

Note: If the function is not found at the command check, it will ignore.

Parameters **func** – Any callable. Coroutines are supported.

class discord_slash.model.SubcommandObject(*sub, base, name, sub_group=None*)

Bases: [discord_slash.model.CommandObject](#)

Subcommand object of this extension.

Note: This model inherits [model.CommandObject](#), so this has every variables from that.

Warning: Do not manually init this model.

Variables

- **base** – Name of the base slash command.
- **subcommand_group** – Name of the subcommand group. None if not exist.
- **base_description** – Description of the base command.
- **subcommand_group_description** – Description of the subcommand_group.

add_check(*func*)

Adds check to the command.

Parameters **func** – Any callable. Coroutines are supported.

async can_run(*ctx*) → bool

Whether the command can be run.

Parameters **ctx** ([context.SlashContext](#)) – SlashContext for the check running.

Returns bool

async invoke(**args*)

Invokes the command.

Parameters **args** – Args for the command.

Raises `.error.CheckFailure`

remove_check(*func*)

Removes check to the command.

Note: If the function is not found at the command check, it will ignore.

Parameters **func** – Any callable. Coroutines are supported.

class discord_slash.model.CogBaseCommandObject(**args*)

Bases: [discord_slash.model.BaseCommandObject](#)

Slash command object but for Cog.

Warning: Do not manually init this model.

async invoke(**args*, ***kwargs*)

Invokes the command.

Parameters **args** – Args for the command.

Raises `.error.CheckFailure`

add_check(*func*)

Adds check to the command.

Parameters **func** – Any callable. Coroutines are supported.

async can_run(*ctx*) → bool

Whether the command can be run.

Parameters **ctx** ([context.SlashContext](#)) – SlashContext for the check running.

Returns bool

remove_check(*func*)

Removes check to the command.

Note: If the function is not found at the command check, it will ignore.

Parameters **func** – Any callable. Coroutines are supported.

class discord_slash.model.CogSubcommandObject(*base, cmd, sub_group, name, sub*)

Bases: [discord_slash.model.SubcommandObject](#)

Subcommand object but for Cog.

Warning: Do not manually init this model.

async invoke(*args, **kwargs)

Invokes the command.

Parameters **args** – Args for the command.

Raises .error.CheckFailure

add_check(func)

Adds check to the command.

Parameters **func** – Any callable. Coroutines are supported.

async can_run(ctx) → bool

Whether the command can be run.

Parameters **ctx** ([context.SlashContext](#)) – SlashContext for the check running.

Returns bool

remove_check(func)

Removes check to the command.

Note: If the function is not found at the command check, it will ignore.

Parameters **func** – Any callable. Coroutines are supported.

class discord_slash.model.SlashCommandOptionType(*value*)

Bases: [enum.IntEnum](#)

Equivalent of [ApplicationCommandOptionType](#) in the Discord API.

SUB_COMMAND = 1

SUB_COMMAND_GROUP = 2

STRING = 3

INTEGER = 4

BOOLEAN = 5

USER = 6

CHANNEL = 7

ROLE = 8

classmethod from_type(*t: type*)

Get a specific SlashCommandOptionType from a type (or object).

Parameters **t** – The type or object to get a SlashCommandOptionType for.

Returns [model.SlashCommandOptionType](#) or None

```
class discord_slash.model.SlashMessage(*, state, channel, data, _http:
    discord_slash.http.SlashCommandRequest, interaction_token)
    Bases: discord.message.Message
    discord.py's discord.Message but overridden edit and delete to work for slash command.
    async edit(**fields)
        Refer discord.Message.edit().
    async delete(*, delay=None)
        Refer discord.Message.delete().

class discord_slash.model.PermissionData(id, type, permission, **kwargs)
    Bases: object
    Single slash permission data.

    Variables
    • id – User or role id, based on following type specific.
    • type – The SlashCommandPermissionsType type of this permission.
    • permission – State of permission. True to allow, False to disallow.

class discord_slash.model.GuildPermissionsData(id, guild_id, permissions, **kwargs)
    Bases: object
    Slash permissions data for a command in a guild.

    Variables
    • id – Command id, provided by discord.
    • guild_id – Guild id that the permissions are in.
    • permissions – List of permissions dict.

class discord_slash.model.SlashCommandPermissionType(value)
    Bases: enum.IntEnum
    Equivalent of ApplicationCommandPermissionType in the Discord API.
    ROLE = 1
    USER = 2
    classmethod from_type(t: type)
```

4.2 Module contents

4.2.1 discord-py-slash-command

Simple Discord Slash Command extension for discord.py

copyright

(c) 2020-2021 eunwoo1104

license MIT

DISCORD_SLASH EVENTS

This page is about events of this extension. These events can be registered to discord.py's listener or event decorator.

on_slash_command(*ctx*)

Called when slash command is triggered.

Parameters **ctx** (`model.SlashContext`) – SlashContext of the triggered command.

on_slash_command_error(*ctx, ex*)

Called when slash command had an exception while the command was invoked.

Parameters

- **ctx** (`model.SlashContext`) – SlashContext of the triggered command.
- **ex** (*Exception*) – Exception that raised.

DISCORD_SLASH.UTILS PACKAGE

6.1 Submodules

6.1.1 discord_slash.utils.manage_commands module

async discord_slash.utils.manage_commands.add_slash_command(*bot_id*, *bot_token*: *str*, *guild_id*,
cmd_name: *str*, *description*: *str*,
options: *Optional[list]* = *None*)

A coroutine that sends a slash command add request to Discord API.

Parameters

- **bot_id** – User ID of the bot.
- **bot_token** – Token of the bot.
- **guild_id** – ID of the guild to add command. Pass *None* to add global command.
- **cmd_name** – Name of the command. Must be 3 or longer and 32 or shorter.
- **description** – Description of the command.
- **options** – List of the function.

Returns JSON Response of the request.

Raises [*error.RequestFailure*](#) - Requesting to Discord API has failed.

async discord_slash.utils.manage_commands.remove_slash_command(*bot_id*, *bot_token*, *guild_id*,
cmd_id)

A coroutine that sends a slash command remove request to Discord API.

Parameters

- **bot_id** – User ID of the bot.
- **bot_token** – Token of the bot.
- **guild_id** – ID of the guild to remove command. Pass *None* to remove global command.
- **cmd_id** – ID of the command.

Returns Response code of the request.

Raises [*error.RequestFailure*](#) - Requesting to Discord API has failed.

async discord_slash.utils.manage_commands.get_all_commands(*bot_id*, *bot_token*, *guild_id*=*None*)

A coroutine that sends a slash command get request to Discord API.

Parameters

- **bot_id** – User ID of the bot.
- **bot_token** – Token of the bot.
- **guild_id** – ID of the guild to get commands. Pass *None* to get all global commands.

Returns JSON Response of the request.

Raises *error.RequestFailure* - Requesting to Discord API has failed.

async discord_slash.utils.manage_commands.**remove_all_commands**(*bot_id, bot_token, guild_ids:*
Optional[List[int]] = None)

Remove all slash commands.

Parameters

- **bot_id** – User ID of the bot.
- **bot_token** – Token of the bot.
- **guild_ids** – List of the guild ID to remove commands. Pass *None* to remove only the global commands.

async discord_slash.utils.manage_commands.**remove_all_commands_in**(*bot_id, bot_token,*
guild_id=None)

Remove all slash commands in area.

Parameters

- **bot_id** – User ID of the bot.
- **bot_token** – Token of the bot.
- **guild_id** – ID of the guild to remove commands. Pass *None* to remove all global commands.

async discord_slash.utils.manage_commands.**get_all_guild_commands_permissions**(*bot_id,*
bot_token,
guild_id)

A coroutine that sends a slash command get request to Discord API.

Parameters

- **bot_id** – User ID of the bot.
- **bot_token** – Token of the bot.
- **guild_id** – ID of the guild to get permissions.

Returns JSON Response of the request. A list of <<https://discord.com/developers/docs/interactions/slash-commands#get-application-command-permissions>>.

Raises *error.RequestFailure* - Requesting to Discord API has failed.

async discord_slash.utils.manage_commands.**update_single_command_permissions**(*bot_id,*
bot_token,
guild_id,
command_id,
permissions)

A coroutine that sends a slash command put request to Discord API.

Parameters

- **bot_id** – User ID of the bot.
- **bot_token** – Token of the bot.
- **guild_id** – ID of the guild to update permissions on.

- **command_id** – ID for the command to update permissions on.
- **permissions** – List of permissions for the command.

Returns JSON Response of the request. A list of <<https://discord.com/developers/docs/interactions/slash-commands#edit-application-command-permissions>>

Raises *error.RequestFailure* - Requesting to Discord API has failed.

```
async discord_slash.utils.manage_commands.update_guild_commands_permissions(bot_id,
                                                                              bot_token,
                                                                              guild_id,
                                                                              cmd_permissions)
```

A coroutine that sends a slash command put request to Discord API.

Parameters

- **bot_id** – User ID of the bot.
- **bot_token** – Token of the bot.
- **guild_id** – ID of the guild to update permissions.
- **permissions** – List of dict with permissions for each commands.

Returns JSON Response of the request. A list of <<https://discord.com/developers/docs/interactions/slash-commands#batch-edit-application-command-permissions>>.

Raises *error.RequestFailure* - Requesting to Discord API has failed.

```
discord_slash.utils.manage_commands.create_option(name: str, description: str, option_type: Union[int,
                                                                                                     type], required: bool, choices: Optional[list] =
                                                                                                     None) → dict
```

Creates option used for creating slash command.

Parameters

- **name** – Name of the option.
- **description** – Description of the option.
- **option_type** – Type of the option.
- **required** – Whether this option is required.
- **choices** – Choices of the option. Can be empty.

Returns dict

Note: An option with `required=False` will not pass anything to the command function if the user doesn't pass that option when invoking the command. You must set the the relevant argument's function to a default argument, eg `argname = None`.

Note: choices must either be a list of `option type dicts` or a list of single string values.

```
discord_slash.utils.manage_commands.generate_options(function: collections.abc.Callable, description:
                                                                                                     str = 'No description.', connector:
                                                                                                     Optional[dict] = None) → list
```

Generates a list of options from the type hints of a command. You currently can type hint: `str`, `int`, `bool`, `discord.User`, `discord.Channel`, `discord.Role`

Warning: This is automatically used if you do not pass any options directly. It is not recommended to use this.

Parameters

- **function** – The function callable of the command.
- **description** – The default argument description.
- **connector** – Kwarg connector of the command.

`discord_slash.utils.manage_commands.create_choice(value: Union[str, int], name: str)`

Creates choices used for creating command option.

Parameters

- **value** – Value of the choice.
- **name** – Name of the choice.

Returns dict

`discord_slash.utils.manage_commands.create_permission(id: int, id_type: Union[int, discord_slash.model.SlashCommandPermissionType], permission: bool)`

Create a single command permission.

Parameters

- **id** – Target id to apply the permission on.
- **id_type** – Type of the id, `model.SlashCommandPermissionsType`.
- **permission** – State of the permission. True to allow access, False to disallow access.

Returns dict

Note: For @everyone permission, set `id_type` as role and `id` as guild id.

`discord_slash.utils.manage_commands.create_multi_ids_permission(ids: List[int], id_type: Union[int, discord_slash.model.SlashCommandPermissionType], permission: bool)`

Creates a list of permissions from list of ids with common `id_type` and permission state.

Parameters

- **ids** – List of target ids to apply the permission on.
- **id_type** – Type of the id.
- **permission** – State of the permission. True to allow access, False to disallow access.

`discord_slash.utils.manage_commands.generate_permissions(allowed_roles: Optional[List[int]] = None, allowed_users: Optional[List[int]] = None, disallowed_roles: Optional[List[int]] = None, disallowed_users: Optional[List[int]] = None)`

Creates a list of permissions.

Parameters

- **allowed_roles** – List of role ids that can access command.
- **allowed_users** – List of user ids that can access command.
- **disallowed_roles** – List of role ids that should not access command.
- **disallowed_users** – List of users ids that should not access command.

Returns list

6.2 Module contents

6.2.1 discord-py-slash-command.utils

Utility functions for slash command.

copyright

(c) 2020-2021 eunwoo1104

license MIT

FREQUENTLY ASKED QUESTIONS

7.1 Why don't my slash commands show up?

If your slash commands don't show up, then you have not added them to Discord correctly. Check these items.

- Ensure that your application has the `applications.commands` scope in that guild.
- If you're creating global command, then you may have to wait up to 1 hour for them to update. It's suggested to use guild command for testing.
- See *How to add slash commands?*.

7.2 How to add slash commands?

Adding a slash command is a two part process.

1. Registering the command to Discord so the command show up when you type `/`.
2. Adding it to your bot.

7.2.1 Add a slash command on Discord

If you want your commands automatically registered, set `sync_commands` to `True` at `client.SlashCommand`.

```
from discord_slash import SlashCommand

slash = SlashCommand(client, sync_commands=True)
```

Or, if you prefer to have more control, you can use `utils.manage_commands`.

Or, you can make requests directly to Discord API, read the [docs](#).

7.2.2 Add the command to your bot

To add a slash command to your bot, you need to use the decorator on a coroutine, just like discord.py's command system but a bit different.

See [Quickstart](#) for an example.

For normal slash command, use `client.SlashCommand.slash()`, and for subcommand, use `client.SlashCommand.subcommand()`.

7.3 How to delete slash commands?

If `sync_commands` is set to `True`, commands will automatically be removed as needed.

However, if you are not using `sync_commands` you can do it manually by this methods:

- Deleting a single command with `utils.manage_commands.remove_slash_command()`
- Deleting all commands using `utils.manage_commands.remove_all_commands()`
- Deleting all commands in a specified guild, or all global commands by `utils.manage_commands.remove_all_commands_in()`
- Making a HTTP request to [discord](#)

To delete a single command yourself you'll have to have the command id, which can be found by getting all commands for a guild / global commands.

7.4 What is the difference between `ctx.send` and `ctx.channel.send`?

- Also answers: How to send files?, How to get `discord.Message` object from `.send()`?, etc.

7.4.1 `ctx.send`

This sends a message or response of the slash command via the interaction response or interaction webhook.

These are only available by this:

1. Show command triggering message. (a.k.a. ACK)
2. Hide the response so only message author can see.

However, you are unable to use after 15 mins from the invocation.

7.4.2 `ctx.channel.send`

`ctx.channel` is the `discord.TextChannel` object for the channel that the slash command was used in. `send()` is the sending coroutine in discord.py. (`discord.TextChannel.send()`)

Warning:

- If the bot is not in the guild, but slash commands are, `ctx.channel` will be `None` and this won't work.
- If the bot does not have view/send permissions in that channel this also won't work, but slash commands show up no matter what the channel specific permissions.

These are not supported compared to `ctx.send`:

1. Showing command triggering message.
2. Hiding the response so only message author can see.

You can use them both together, just be aware of permissions.

7.5 Can I use something of discord.py's in this extension?

Most things work, but a few that are listed below don't.

7.5.1 Checks

discord.py check decorators can work, but its not 100% guaranteed every checks will work.

7.5.2 Events

Command-related events like `on_command_error`, `on_command`, etc. This extension triggers some events, check the [events docs](#)

7.5.3 Converters

Use `options=[...]` on the slash command / subcommand decorator instead.

Note: Pretty much anything from the discord's commands extension doesn't work, also some bot things.

Warning: If you use something that might take a while, eg `wait_for` you'll run into two issues:

1. If you don't respond within 3 seconds (`ctx.defer()` or `ctx.send(..)`) discord invalidates the interaction.
2. The interaction only lasts for 15 minutes, so if you try and send something with the interaction (`ctx.send`) more than 15 mins after the command was ran it won't work.

As an alternative you can use `ctx.channel.send` but this relies on the the bot being in the guild, and the bot having send perms in that channel.

7.6 Any more questions?

Join the [discord server](#) and ask in `#questions`!

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

d

- `discord_slash`, [33](#)
- `discord_slash.client`, [15](#)
- `discord_slash.cog_ext`, [21](#)
- `discord_slash.context`, [23](#)
- `discord_slash.error`, [25](#)
- `discord_slash.http`, [26](#)
- `discord_slash.model`, [28](#)
- `discord_slash.utils`, [41](#)
- `discord_slash.utils.manage_commands`, [37](#)

INDEX

A

`add_check()` (*discord_slash.model.BaseCommandObject* method), 30
`add_check()` (*discord_slash.model.CogBaseCommandObject* method), 31
`add_check()` (*discord_slash.model.CogSubcommandObject* method), 32
`add_check()` (*discord_slash.model.CommandObject* method), 29
`add_check()` (*discord_slash.model.SubcommandObject* method), 30
`add_slash_command()` (*discord_slash.client.SlashCommand* method), 16
`add_slash_command()` (*discord_slash.http.SlashCommandRequest* method), 26
`add_slash_command()` (in module *discord_slash.utils.manage_commands*), 37
`add_subcommand()` (*discord_slash.client.SlashCommand* method), 17
`AlreadyResponded`, 25
`application_id` (*discord_slash.http.SlashCommandRequest* property), 26

B

`BASE` (*discord_slash.http.CustomRoute* attribute), 26
`BaseCommandObject` (class in *discord_slash.model*), 29
`BOOLEAN` (*discord_slash.model.SlashCommandOptionType* attribute), 32
built-in function
 `on_slash_command()`, 35
 `on_slash_command_error()`, 35

C

`can_run()` (*discord_slash.model.BaseCommandObject* method), 30
`can_run()` (*discord_slash.model.CogBaseCommandObject* method), 31

`can_run()` (*discord_slash.model.CogSubcommandObject* method), 32
`can_run()` (*discord_slash.model.CommandObject* method), 29
`can_run()` (*discord_slash.model.SubcommandObject* method), 31
`channel` (*discord_slash.context.SlashContext* property), 24
`CHANNEL` (*discord_slash.model.SlashCommandOptionType* attribute), 32
`CheckFailure`, 25
`ChoiceData` (class in *discord_slash.model*), 28
`cog_slash()` (in module *discord_slash.cog_ext*), 21
`cog_subcommand()` (in module *discord_slash.cog_ext*), 22
`CogBaseCommandObject` (class in *discord_slash.model*), 31
`CogSubcommandObject` (class in *discord_slash.model*), 32
`command_request()` (*discord_slash.http.SlashCommandRequest* method), 26
`command_response()` (*discord_slash.http.SlashCommandRequest* method), 27
`CommandData` (class in *discord_slash.model*), 28
`CommandObject` (class in *discord_slash.model*), 28
`create_choice()` (in module *discord_slash.utils.manage_commands*), 40
`create_multi_ids_permission()` (in module *discord_slash.utils.manage_commands*), 40
`create_option()` (in module *discord_slash.utils.manage_commands*), 39
`create_permission()` (in module *discord_slash.utils.manage_commands*), 40
`CustomRoute` (class in *discord_slash.http*), 26

D

`defer()` (*discord_slash.context.SlashContext* method), 24
`deferred` (*discord_slash.context.SlashContext* property), 23

`delete()` (*discord_slash.http.SlashCommandRequest* method), 28

`delete()` (*discord_slash.model.SlashMessage* method), 33

`discord_slash`
module, 33

`discord_slash.client`
module, 15

`discord_slash.cog_ext`
module, 21

`discord_slash.context`
module, 23

`discord_slash.error`
module, 25

`discord_slash.http`
module, 26

`discord_slash.model`
module, 28

`discord_slash.utils`
module, 41

`discord_slash.utils.manage_commands`
module, 37

`DuplicateCommand`, 25

`DuplicateSlashClient`, 25

E

`edit()` (*discord_slash.http.SlashCommandRequest* method), 27

`edit()` (*discord_slash.model.SlashMessage* method), 33

F

`from_type()` (*discord_slash.model.SlashCommandOptionType* class method), 32

`from_type()` (*discord_slash.model.SlashCommandPermissionType* class method), 33

G

`generate_options()` (in module *discord_slash.utils.manage_commands*), 39

`generate_permissions()` (in module *discord_slash.utils.manage_commands*), 40

`get_all_commands()` (*discord_slash.http.SlashCommandRequest* method), 26

`get_all_commands()` (in module *discord_slash.utils.manage_commands*), 37

`get_all_guild_commands_permissions()` (*discord_slash.http.SlashCommandRequest* method), 26

`get_all_guild_commands_permissions()` (in module *discord_slash.utils.manage_commands*), 38

`get_cog_commands()` (*discord_slash.client.SlashCommand* method), 16

`guild` (*discord_slash.context.SlashContext* property), 23

`GuildPermissionsData` (class in *discord_slash.model*), 33

H

`handle_subcommand()` (*discord_slash.client.SlashCommand* method), 20

I

`IncorrectCommandData`, 25

`IncorrectFormat`, 25

`IncorrectType`, 25

`INTEGER` (*discord_slash.model.SlashCommandOptionType* attribute), 32

`invoke()` (*discord_slash.model.BaseCommandObject* method), 30

`invoke()` (*discord_slash.model.CogBaseCommandObject* method), 31

`invoke()` (*discord_slash.model.CogSubcommandObject* method), 32

`invoke()` (*discord_slash.model.CommandObject* method), 29

`invoke()` (*discord_slash.model.SubcommandObject* method), 31

`invoke_command()` (*discord_slash.client.SlashCommand* method), 20

M

module

- `discord_slash`, 33
- `discord_slash.client`, 15
- `discord_slash.cog_ext`, 21
- `discord_slash.context`, 23
- `discord_slash.error`, 25
- `discord_slash.http`, 26
- `discord_slash.model`, 28
- `discord_slash.utils`, 41
- `discord_slash.utils.manage_commands`, 37

O

`on_slash_command()`
built-in function, 35

`on_slash_command_error()`
built-in function, 35

`on_slash_command_error()` (*discord_slash.client.SlashCommand* method), 21

`on_socket_response()` (*discord_slash.client.SlashCommand* method), 20

`OptionData` (class in *discord_slash.model*), 28

P

`permission()` (*discord_slash.client.SlashCommand* method), 20

`PermissionData` (class in *discord_slash.model*), 33

`post_followup()` (*discord_slash.http.SlashCommandRequest* method), 27

`post_initial_response()` (*discord_slash.http.SlashCommandRequest* method), 27

`process_options()` (*discord_slash.client.SlashCommand* method), 20

`put_slash_commands()` (*discord_slash.http.SlashCommandRequest* method), 26

R

`remove_all_commands()` (in module *discord_slash.utils.manage_commands*), 38

`remove_all_commands_in()` (in module *discord_slash.utils.manage_commands*), 38

`remove_check()` (*discord_slash.model.BaseCommandObject* method), 30

`remove_check()` (*discord_slash.model.CogBaseCommandObject* method), 31

`remove_check()` (*discord_slash.model.CogSubcommandObject* method), 32

`remove_check()` (*discord_slash.model.CommandObject* method), 29

`remove_check()` (*discord_slash.model.SubcommandObject* method), 31

`remove_cog_commands()` (*discord_slash.client.SlashCommand* method), 16

`remove_slash_command()` (*discord_slash.http.SlashCommandRequest* method), 26

`remove_slash_command()` (in module *discord_slash.utils.manage_commands*), 37

`request_with_files()` (*discord_slash.http.SlashCommandRequest* method), 27

`RequestFailure`, 25

`ROLE` (*discord_slash.model.SlashCommandOptionType* attribute), 32

`ROLE` (*discord_slash.model.SlashCommandPermissionType* attribute), 33

S

`send()` (*discord_slash.context.SlashContext* method), 24

`slash()` (*discord_slash.client.SlashCommand* method), 18

`SlashCommand` (class in *discord_slash.client*), 15

`SlashCommandError`, 25

`SlashCommandOptionType` (class in *discord_slash.model*), 32

`SlashCommandPermissionType` (class in *discord_slash.model*), 33

`SlashCommandRequest` (class in *discord_slash.http*), 26

`SlashContext` (class in *discord_slash.context*), 23

`SlashMessage` (class in *discord_slash.model*), 32

`STRING` (*discord_slash.model.SlashCommandOptionType* attribute), 32

`SUB_COMMAND` (*discord_slash.model.SlashCommandOptionType* attribute), 32

`SUB_COMMAND_GROUP` (*discord_slash.model.SlashCommandOptionType* attribute), 32

`subcommand()` (*discord_slash.client.SlashCommand* method), 19

`SubcommandObject` (class in *discord_slash.model*), 30

`sync_all_commands()` (*discord_slash.client.SlashCommand* method), 16

T

`to_dict()` (*discord_slash.client.SlashCommand* method), 16

U

`update_guild_commands_permissions()` (*discord_slash.http.SlashCommandRequest* method), 26

`update_guild_commands_permissions()` (in module *discord_slash.utils.manage_commands*), 39

`update_single_command_permissions()` (in module *discord_slash.utils.manage_commands*), 38

`USER` (*discord_slash.model.SlashCommandOptionType* attribute), 32

`USER` (*discord_slash.model.SlashCommandPermissionType* attribute), 33