Piscord

Jul 01, 2020

Table of content

CHAPTER 1

Tutorial Piscord

1.1 Introduction

1.1.1 Que permet piscord ?

Piscord permet, au même titre que Discord.py, de faire des bots discord. Seulement, l'avantage est que Piscord simplifie la création de bots sur plusieurs points.

Tout d'abord, la programmation avec Piscord est synchrone, ce qui en simplifie l'utilisation, surtout par des personnes pas forcément expertes en python

Ensuite, la librairie n'est pas forcément bloquante. Cela permet de faire plusieurs bots en un script, de lancer un serveur web en même temps.. A peu près ce que l'on veut.

Enfin, nous essayons de faire que ce soit le plus simple possible de faire un dialogue avec le web. Il y a notamment une classe (OAuth) qui est la pour permettre d'utiliser la connexion avec discord facilement en faisant un site, et d'exploiter ces données comme un bot le ferais.

1.1.2 Quels sont les prérequis pour l'utiliser ?

Les prérequis demandés pour utiliser piscord sont simples

- Savoir lire une doc (ce que vous êtes en train de faire)
- Connaitres les bases (variables, boucles, comparaison, fonctions)
- Savoir utiliser des classes
- Savoir débugguer

De plus, connaitre les décorateurs est utile mais pas indispensable vu qu'ils sont utilisés partout mais l'on ne s'attarde pas sur leur fonctionnement.

1.1.3 Comment installer piscord ?

Pour installer la librairie, cela est très simple, il suffit de l'installer avec pip, (pip install -U piscord). Cela va installer les dépendances de cette dernière, pour vous permettre de l'utiliser directement après.

Ensuite, dans votre code python, utilisez *import piscord*, *from piscord import ** ou *from piscord import things* selon comment vous voulez l'utiliser.

1.2 First Bot

1.2.1 Mettre en ligne le bot

Pour mettre en ligne votre premier bot, récupérez le **token** de ce dernier et faites comme ceci :

```
from piscord import Bot
# Import de la classe Bot a partir de la librairie

bot = Bot("Token")
# Création d'un Bot avec votre token

@bot.event
def on_ready(ready):
        print(f"Bot {ready.user.name} Connected")
# Créer un event correpondant à celui de "on_ready" (quand le bot est connecté a␣
↪discord), et dire que le bot est lancé.

bot.run()
# Lancer le bot en mode bloquant
```

Information :

Vous avez deux façons de créer un event : - En utilisant @bot.event et en renommant la fonction du nom de l'event - En utilisant @bot.event("nom_de_l_event") et en renommant la fonction comme vous le souhaitez

Ce qui donne ça :

```
@bot.event
def on_ready(ready):...

ou

@bot.event("on_ready")
def ready(ready):...
```

Aussi, vous pouvez lancer le bot en mode bloquant (le code après ne sera pas exécuté) ou non bloquant.

```
bot.run()
# Manière bloquante, a utiliser si l'on ne compte rien faire en même temps que lancer␣
↪le bot.

bot.start()
# Manière non bloquante, si on lance d'autres bots après, ou un serveur web en␣
↪parallèle.
```

Ping Pong

Une manière souvent utilisé pour illustrer la création d'un pemier bot est une commande ("ping") qui fera répondre une autre ("pong") au bot. Voyons comment le faire avec Piscord :

```
from piscord import Bot

bot = Bot("Token")

@bot.event
def on_ready(ready):
        print(f"Bot {ready.user.name} Connected")

@bot.event
def on_message(message):
        if message.content == "!ping":
                message.channel.send("Pong !")
# Quand un message est envoyé, on vérifie si son contenu est "!ping".
# Si c'est le cas, on envoi dans le salon le message "Pong !"

bot.run()
```

1.2.2 Objet Message

Quand on déclare un event, on récupère en argument un objet Event nous permettant de récupérer des informations de ce dernier.

Pour les voir je vous invite a aller consulter [La Documentation au sujet des Messages](https://piscord.astremy.com/#Message). Sinon, voyons ensemble les plus utiles :

```
Message.content
# Le contenu du message, ce qu'à saisi l'utilisateur. Cela permet d'identifier des␣
↪éventuels commandes, gros mots...
# C'est le message en lui-même (une chaine de caractères).

Message.author
# L'auteur du message. Un objet User se rapportant à l'utilsateur qui a éxécuté la␣
↪commande,
# permettant d'avoir diverses informations sur lui (pseudo, id..)

Message.channel
# Le salon dans lequel a été envoyé le message. Permet d'y renvoyer un message,
# Ou de récupérer des informations sur le salon.

Message.guild
# Le serveur dans lequel a été envoyé le message.
# Permet d'en récupérer des informations.
```

1.2.3 Un bot basique

Voici un exemple un peu plus développé de l'utilisation de l'argument (je ne montre pas tout le code, seulement la partie event) :

```
@bot.event
def on_message(message):
        if message.content == "!infos":
                if message.guild:
                        server = message.guild.name
                else:
                        server = "Aucun, nous sommes en messages privés"
                message.channel.send(f"Informations :\nUtilisateur : {message.author}
↪\nServeur : {server}")
```

1.3 Envoyer des messages

Pour envoyer un message, il y a deux façons de s'y prendre

```
channel.send(message)
# Façon largement préférée, facile d'utilisation et simple à comprendre.

bot.send_message(channel_id, message)
# Ancienne forme, dépréciée, a utiliser le moins possible, sauf dans des cas très␣
↪précis.
```

Dans ce tutoriel, on s'attardera sur la première forme.

Le premier argument de channel.send() est le contenu (content) du message. Il n'est pas obligé d'être spécifié en tant que kwarg (sous forme *Channel.send(content = "contenu")*) mais peut simplement être mis directement *Channel.send("contenu")*.

Cependant, pour le reste des arguments, il faut spécifier le nom.

1.3.1 Arguments

Il y a différents arguments que l'on peut mettre dans le send :

- tts : Une valeur True ou False, si le message envoyé est un text-to-speech.
- files : Une liste des nom de fichiers que l'on souhaite envoyer (si l'on en envoie).
- embed : Un embed, nous verrons plus loin comment en faire.
- allowed_mentions : Un objet Allowed_Mentions, nous verrons également comment le faire plus loin.

Allowed_Mentions

Les mentions autorisés permettent d'empêcher que le bot mentionne par mégarde quelque-chose qu'il ne devrait pas (ex : mentionner everyone parce qu'il a les perms et qu'un malin lui fait répéter une mention qu'il ne peut pas utiliser).

Il a plusieurs paramètres : Parse : Parse est essentiel quand on joue avec la classe. Elle indique les types de mentions a autoriser, même si on doit les détailler après. C'est une liste qui peut prendre les arguments que l'on veut selon ce que l'on souhaite faire.

- "everyone" : Autorise les mentions d'*@everyone* et *@here*
- "users" : Permet de mentionner les utilisateurs.
- "roles" : Permet de mentionner les rôles.

Ainsi, par exemple, vous pouvez faire :

```
Allowed_Mentions.parse = []
# Supprime toutes les mentions

Allowed_Mentions.parse = ["everyone"]
# Ne permet au bot de mentionner seulement everyone/here (s'il en as les permissions)
```

Users : Users est un paramètre de la classe qui, si on ne permet pas de mentionner tous les utilisateurs (en mettant "users"), permet de quand même en mentionner, mais en précisant les id des utilisateurs a laisser (avec un maximum de 100).

Roles : Roles est comme Users, mais pour les rôles. Si on ne permet pas au bot de mentionner tous les rôles, permet de spécifier l'id des rôles a pouvoir mentionner quand même.

Exemple

Voici un exemple de commande que l'on peut faire, ou cela se trouve utile :

```
@bot.event
def on_message(message):
        content = message.content.split()
        if content[0] == "!me":
                if len(content) > 1:
                        allow = Allowed_Mentions() # ou Allowed_Mentions({}) selon la␣
↪version.
                        allow.parse = []
                        allow.users = [message.author.id]
                        message.channel.send(" ".join(content[1:]),allowed_
↪mentions=allow.to_json())

# Envoie le message que l'utilisateur a mis après la commande !me, tout en ne pouvant␣
↪mentionner que l'auteur de la commande.
```

1.3.2 Les embeds

Les embeds sont quelque chose de très importants par rapport au messages, et mérite que l'on s'y attarde.

Ces derniers ont de nombreuses propriétés, et l'on s'attardera pas sur toutes. Pour plus d'informations, visitez [La Documentation relative aux Embed](https://piscord.astremy.com/#Embed) ou [Les informations de Discord sur les Embed](https://discord.com/developers/docs/resources/channel#embed-object).

Voici les plus utiles :

- title : Une chaine de caractère correspondante au titre de l'Embed.
- description : Le texte dans l'Embed.
- color : La couleur de l'Embed (sous format hexadécimal passé en décimal).
- Image : Un objet image correspondant à une image principale de l'Embed. (Voir la documentation)

De plus, les Embed ont ce que l'on appelle les fields. Ce sont des zones qui contiennent chacun leur titre et texte que l'on met dans un Embed. Pour ajouter un field a un embed, on utilise Embed.add_field(name = name, value = value, inline = inline)

Exemple

```
embed = Embed() # Crée l'embed

embed.title = "Description des commandes" # Défini le titre de l'embed
embed.color = 3375070 # Défini la couleur

embed.add_field(name="!me",value="Commande qui répète du texte, en ne pouvant␣
↪mentionner que soit-même")
# Le name correspond au titre du field, la value à son contenu. Mettre inline = True␣
↪permet de faire revenir le bloc a la ligne.
embed.add_field(name="!ping",value="Commande de base qui répond 'Pong !'")

Channel.send(embed=embed.to_json())
```

Quand l'on envoie un embed dans un channel, il ne faut pas oublier de mettre un .to_json, comme pour le Allowed_Mentions.

1.4 Les Permissions

1.4.1 Comment fonctionnent les permissions ?

Sur discord, il existe plusieurs niveaux de permissions donnant accès aux utilisateurs à certaines fonctionnalitées. Par exemple, prennons la permission `KICK_MEMBERS` donnera le droit à l'utilisateur de kick un membre du serveur actuel. Mais comment peut-on par exemple, vérifier les permissions que possède un membre ?

```
from piscord import Bot, Permission

bot = Bot("Token")

@bot.event
def on_message(message):
        everyone = message.guild.roles[0]
        # On récupère le rôle @everyone

        print(everyone.permissions)
        # On affiche dans la console les permissions du rôle

bot.run()
```

Comme vous pouvez le remarquer si vous testez le bout de code plus haut, on ne reçois dans la console qu'un simple nombre. En fait, ce nombre correspond à la valeur décimal des permissions qui sont exprimées en binaire. Comment cette fois si, vérifier si une personne possède une permission en particulier ? Reprenons le code plus haut :

```
@bot.event
def on_message(message):
        everyone = message.guild.roles[0]
        # On récupère le rôle @everyone

        if everyone.permissions == Permission.SPEAK:
                # On vérifie si la permissions accordée au rôle everyone est bien␣
↪celle de parler

                print("Le rôle everyone peut bien parler dans les channels vocaux")
```

Exemple d'utilisation

Voici un exemple de commande utilisant les permissions, une commande pour kick :

```
@bot.event
def on_message(message):
        mes = message.content.split()
        # On récupère le contenu du message, qu'on sépare en liste de mot

        if mes[0] == "!kick":
                perm = False

                for role in message.author.roles:
                # On parcourt la liste des rôles du membre
                        if role.permissions in (Permission.KICK_MEMBERS, Permission.
↪ADMINISTRATORS):
                        # On vérifie si le rôle en question à les perms pour kick␣
↪(admin ou kick)

                                perm = True
                                break

                if perm:
                        if len(message.mentions):
                                # On vérifie si un user à été mentionné

                                member = message.mentions[0]
                                bot.get_element(message.guild.members, id=member.id).
↪kick()
                                # On récupère l'objet Member correspondant à la␣
↪mention et on kick le membre

                                message.channel.send(f"{message.mentions[0]} has been␣
↪kicked")
                        else:
                                message.channel.send("You have to mentions a member")
                else:
                        message.channel.send("You do not have the permissions")
```

Quelques opérateurs sur les Permissions

Comme vu dans l'exemple précédent, il existe plusieurs opérateur permettant de faire des vérifications de permissions. Il en existe actuellement 3. Le +, le −, et le ==. Reprenons l'exemple plus haut :

```
if role.permissions == Permission.KICK_MEMBERS + Permission.ADMINISTRATORS:
        ...
```

Ici, grâce à l'opérateur + et ==, on peut vérifier si le les permissions du role voulu sont bien `KICK_MEMBERS` et `ADMINISTRATORS`. Le dernier opérateur peut être utile dans le cas des Overwrite par exemple, que nous verrons juste après. Ainsi, le + rajoute une permission si elle n'existe pas, le − l'enlève si elle existe, et le == vérifie si le rôle a les permissions. Aussi, on peut avoir des variantes comme le += qui permet d'ajouter la permission en la réaffectant. On peut également vérifier si un rôle a une permissions dans une liste de permissions donné avec le mot clé `in`. Par exemple :

```
if role.permissions in (Permission.KICK_MEMBERS, Permission.ADMINISTRATORS):
        ...
```

Ici, on va vérifier si le rôle voulu possède au moins la permission `KICK_MEMBERS` ou `ADMINISTRATORS`.

1.4.2 Les Overwrites

L'objet Overwrite permet, comme son nom l'indique, de réecrire les permissions de quelque chose, comme un membre, un rôle, ou un channel. Nous resterons ici sur le cas des channels. L'objet Channel possède un attribut permettant de récupérer les Overwrite de ce channel. Cet attribut est `Channel.permission_overwrite`, et il retourne une liste d'un élément, étant les Overwrite de ce dit channel. Ainsi, on peut récupérer les permissions que possède le rôle @everyone d'un channel, et les modifier. Voyons un exemple. Admettons que nous voulons enlever la permission au rôle @everyone de parler sur un channel voulu

```
# Admettons être dans l'event on_message

perms = message.channel.permission_overwrites[0]
# On récupère l'Overwrite du channel

allow = perms.allow - Permission.SEND_MESSAGE
# On définit les permissions autorisées, ici toutes les permissions du channel, en␣
↪enlevant celle de parler

deny = perms.deny + Permission.SEND_MESSAGE
# On définit les permissions interdites, ici toutes les permissions du channel, en␣
↪ajoutant celle de parler

perms.edit(allow=allow, deny=deny)
# Enfin, on applique les changement au channel en utilisant la méthode edit de l
↪'objet Overwrite
```

Ici, dans la variable `allow`, on peut voir que l'on fait une soustraction de 2 permissions. Ainsi, on récupère la valeur des permissions du rôle @everyone, et on y soustrait la permission `Permission.SEND_MESSAGES` pour enfin appliquer les changements.

1.5 Les Commandes

1.5.1 Présentation du Handler

Par défaut, on crée des commandes avec piscord en détectant un message spécifique, comme !ping ou n'importe quoi du genre. Seulement, il y a un outil pour simplifier la création de commandes : Le Handler. Le Handler est un outil qui va récupérer l'event on_message et l'utiliser pour y trouver la commande que l'on souhaite et la rediriger automatiquement vers la fonction correspondante.

Comment l'utilise-t-on ?

```
from piscord import Handler

bot = Handler("Token","Prefix")

@bot.command
def ping(message):
        message.channel.send("Pong !")

bot.run()
```

Que fait ce bout de code ?

Il crée un bot qui est géré par le Handler, et avec un certain prefix, ce qui va permettre de créer des commandes, ce qu'il fait juste en-dessous avec le @bot.command. Le nom de la fonction après est le nom de la commande. Cela sera détecter quand on fera la commande et renverra à la fonction.

Par exemple, si le prefix est "!", le bot réagira à !ping.

1.5.2 Load et Unload de Commandes

Vous pouvez mettre vos commandes dans des fichiers exterieur et les load et unload a volonté.

Pour cela, mettez tout vos fichiers de commande dans un dossier `commands/`. Ensuite, dans votre fichier de bot, vous avez juste a utiliser `bot.load_module("Name")` et `bot.unload_module("Name")`

Voici ce que cela pourrais donner:

```
from piscord import Handler

bot = Handler("Token","Prefix")

@bot.event
def on_ready(ctx):
        bot.load_module("moderation") # Load le fichier présent à commands/moderation.
↪py
        bot.load_module("economy") # Load le fichier présent à commands/economy.py

@bot.command
def unload_mod(message):
        bot.unload_module("moderation")

bot.run()
```

Caution: Petites précisions

Vous ne pouvez pas définir d'event ou load/unload un module dans un fichier. De plus, utilisez dedans `bot` pour définir vos commandes, c'est celui mis en place pour que la commande puisse bien s'unload.

CHAPTER 2

Piscord Objects

2.1 Bot

class piscord.**Bot_Element**(*bot_element*, *bot*)

Represent the Bot

user: ***User*** The user corresponding to the bot (name, id, avatar)

guilds: ***Guild*** List of guilds where the bot is

relationships: List of relationships of the bot (useless for real bot)

private_channels: ***Channel*** List of private channels of the bot

presences: Not implemented : Presences of the users

voices: List of Voice connexion of the bot

create_guild(***kwargs*)

Create a guild channel, with parameters. Parameters : https://discord.com/developers/docs/resources/guild#create-guild

Return *Guild*

edit(***modifs*)

Modify bot user, with parameters. Parameters : https://discord.com/developers/docs/resources/user#modify-current-user

2.2 Guild

class piscord.**Guild**(*guild*, *bot*)

Represent a discord server

id: ID of the Guild

name: Name of the Guild

icon: The icon (image show on the guilds menu) of the Guild

splash: The background invite image of the Guild

discovery_splash: The background invite image of the Guild in discovery tab

owner: If the user is the owner of the guild (see : https://discord.com/developers/docs/resources/user#get-current-user-guilds)

owner_id: If owner is not specified, the owner id of the Guild

permissions: The permissions in Guild for the user (see : https://discord.com/developers/docs/resources/user#get-current-user-guilds)

region: Not implemented : Voice region of the guild

afk_channel_id: The id of the afk voice channel

afk_timeout: The time before an inactive user is sent to the afk voice channel

embed_enabled: If the server widget is enabled (deprecated, replaced with widget_enabled)

embed_channel_id: The channel id where the widget generate an invite (deprecated, replaced with widget_channel_id)

verification_level:

The level of verification of the server

- 0 : Unrestricted
- 1 : Need email verified
- 2 : Register longer than 5 minutes
- 3 : In the guild longer than 10 minutes
- 4 : Require verified phone number

default_message_notifications:

The level of messages notification by default

- 0 : All messages
- 1 : Mentions Only

explicit_content_filter:

Filter for nsfw content (image)

- 0 : No filter
- 1 : Filter just for roleless members
- 2 : All members

roles: *`Role`* List of Guild roles

emojis: *`Emoji`* List of Guild emojis

features: Not implemented

mfa_level:

If the guild requiere MFA

- 0 : No
- 1 : Yes

application_id: If a bot created the server, the id of its application

widget_enabled: If the server widget is enabled

widget_channel_id: The channel id where the widget generate an invite

system_channel_id: The id of channel system messages (boost, welcome)

system_channel_flags:

A integer representing the system messages enable 0 : All system messages 1 : Boost notification messages 2 : Welcome messages 3 : No system messages

rules_channel_id: The id of rules channel for public guilds

joined_at: The timestamp of guild creation

large: If the guild is considered a large guild

unavailable: If the guild is unavaible

member_count: The number of guild members

voice_states: Not implemented

members: ***Member*** List of guild members

channels: ***Channel*** List of guild channels

presences: Not implemented

max_presences: Max number of presence for the guild, 25000 by default

max_members: Max number of members for the guild

max_video_channel_users: Max number of users in a video channel

vanity_url_code: Custom url for discord parteners and guild level 3

description: Guild description in discover tab

banner: Banner of the guild

premium_tier:

The level of server boosting:

- 0 : Level 0
- 1 : Level 1
- 2 : Level 2
- 3 : Level 3

premium_subscription_count: The number of guild nitro boosts

preferred_locale: The preferred locale of a public guild (for discovery tab)

public_updates_channel_id: The staff channel for Discord notices of a public guild

approximate_member_count: The approximate number of guild members

approximate_presence_count: The approximate number of connected guild members

count_prune(*days=7*, *include_roles=[]*)

Count the number of users will be pruned if you start a prune

days: The number of days the user need to be inactive to be counted

include_roles: The roles to be considered to prune (by default, a user with a role can't be pruned)

create_channel(***kwargs*)
Create a guild channel, with parameters. Parameters : https://discord.com/developers/docs/resources/guild#create-guild-channel

Return *Channel*

create_role(***kwargs*)
Create a guild role, with parameters. Parameters : https://discord.com/developers/docs/resources/guild#create-guild-role

Return *Role*

delete()
Delete permanently the guild. The bot must be the owner

edit(***modifs*)
Modify guild, with parameters. Parameters : https://discord.com/developers/docs/resources/guild#modify-guild

get_ban(*user_id*)
returns a specific *Ban* using the id of the banned user

get_bans()
Return a list of *Ban* of the guild

get_channels()
Return a list of *Channel* of the guild (deprecated, use Guild.channels)

get_invites()
Return a list of *Invite* of the guild

get_member(*user_id*)
returns a specific *Member* using their id

get_members(*limit=100*, *after=0*)
Return a list of *Member* of the guild (deprecated, use Guild.members)

get_roles()
Return a list of *Role* of the guild (deprecated, use Guild.roles)

get_webhooks()
Return a list of *Webhook* of the guild

prune(*days=7*, *include_roles=[]*)
Prune inactive members (kick)

days: The number of days the user need to be inactive to be counted

include_roles: The roles to be considered to prune (by default, a user with a role can't be pruned)

2.3 Channel

class piscord.**Channel**(*channel*, *bot*, *guild=None*)
Represent a discord channel

id: ID of the Channel

type:

The type of the channel 0 : Text channel of a Guild 1 : DM channel 2 : Voice channel of a Guild 3 : DM group channel 4 : Category 5 : News channel 6 : Store channel

guild_id: The guild id of the channel (if is not a dm channel)

position: The channel position in the guild channels

permission_overwrites: The permissions for members and roles in the channel

name: Name of the channel

topic: The channel topic

nsfw: If the channel is or not a nsfw channel

last_message_id: The id of the last channel message

bitrate: The bitrate of the channel (if this is a voice channel)

user_limit: The limit of users in the channel (if this is a voice channel)

rate_limit_per_user: The time between two messages, in seconds

recipients: ***User*** The DM group users

icon: The icon of the DM group

owner_id: The id of the DM group owner

application_id: If a bot created the DM group, the id of its application

parent_id: If the channel is in a category, the category id

last_pin_timestamp: timestamp when the last pinned message was pinned

invites: ***Invite*** List of channel invites

mention: The mention of the channel

guild: ***Guild*** The guild of the channel

bulk_delete(*messages_ids*)
Delete multiple messages

messages_ids: The ids of the message to delete

Max : 100

create_invite(***kwargs*)
Create a guild invite for the channel, with parameters Parameters : https://discord.com/developers/docs/resources/channel#create-channel-invite

Return *Invite*

create_webhook(*name*, *avatar=None*)
Create a webhook for the channel

name: The name of the webhook

avatar: The avatar image data (see : https://discord.com/developers/docs/reference#image-data) of the webhook

Return *Webhook*

edit(***modifs*)
Modify channels, with parameters. Parameters : https://discord.com/developers/docs/resources/channel#modify-channel

get_invites()
Get list of invites of the channel

Return List of *Invite*

get_message(*message_id*)
Get specific message of the channel with it id

Return *Message*

get_messages(*limit=50*, *before=None*, *after=None*)
Get list of messages in the channel

limit: The max number of messages return (max : 100)

before: ID of a message : Retrieves messages that are before the message

after: ID of a message : Retrieves messages that are after the message

Return List of *Message*

get_webhooks()
Get list of webhooks of the channel

Return List of *Webhook*

purge(*max*, *before=None*, *after=None*)
Delete messages from a channels

max: The number of messages to delete

Max : 100

before: The messages before a message id

after: The messages after a message id

send(*content=None*, *files=None*, ***kwargs*)
Send a message in the channels, with parameters. Parameters : https://discord.com/developers/docs/resources/channel#create-message

By default, when not kwarg specified, arg is the content.

Use send(files=[“name_1”,”name_2”]) to send files by their names

Use send(files=[[b”data_2”,”title_1”],[b”data_2”,”title_2”]]) to send files by their data

Return *Message*

typing()
Send a “typing” event in the channel (‘bot typing...’) until the bot sends a message

2.4 Message

class piscord.**Message**(*message*, *bot*)
Represent a message send in a channel by a user

id: ID of the message

channel_id: ID of the channel where the message is

guild_id: ID of the guild where the message is (if is not a DM)

author: ***User*** **or** ***Member*** A user object of the author. If the channel is not a DM, this is a member object

content: The content of the message

timestamp: The timestamp when the message was sent

edited_timestamp: The timestamp when the message was edited

tts: If the message was a TTS message

mention_everyone: If the message mention everyone

mentions: ***User*** List of users mentionned in the message

mention_roles: ***Role*** List of roles mentionned in the message

mention_channels: ***Channel*** List of channels mentionned in the message

(incomplete object, see : https://discord.com/developers/docs/resources/channel#channel-mention-object)

attachments: **`Attachement`** List of attachments of the message

embeds: ***Embed*** If the message have embeds, list of embeds in the message

reactions: ***Reaction*** List of reactions of the message

nonce: Nonce of the message

pinned: If the message is pinned

webhook_id: Webhood id if the message was generated by a webhook

type:

The type of message

- 0 : A normal message
- 1 : A DM group member add
- 2 : A DM group member remove
- 3 : A DM call start
- 4 : A change of DM group channel name
- 5 : A change of DM group channel icon
- 6 : A channel pin
- 7 : A guild member arrival
- 8 : A guild boost
- 9 : A guild boost tier 1
- 10 : A guild boost tier 2
- 11 : A guild boost tier 3
- 12 : A channel follow add
- 13 : A guild discovery disqualified
- 14 : A guild discovery requalified

activity: Not implemented

application: Not implemented

message_reference: Not implemented

flags: Message flags (see : https://discord.com/developers/docs/resources/channel#message-object-message-flags)

guild: ***Guild*** The guild where the message was sent (if is not in DM)

channel: ***Channel*** The channel where the message was sent

add_reaction(*reaction*)
Add the reaction in the message

reaction: A emoji.react string

delete()
Delete the message

delete_reaction(*reaction*, *user_id=None*)
If user_id is specified, delete the reaction of a specific user, else, delete all the reactions corresponding to the reaction in argument

delete_reactions()
Delete all reactions on the message

delete_self_reaction(*reaction*)
Delete its own reaction

edit(***modifs*)
Modify channels, with parameters. Parameters : https://discord.com/developers/docs/resources/channel#edit-message

2.5 User

class piscord.**User**(*user*, *bot*)
Represent a user of discord

id: ID of the user

name: Username of the user

discriminator: Discriminator of the user

avatar: Avatar link of the user (discord cdn)

bot: If the user is a bot

system: If the user is a Official Discord System user

mfa_enabled: If the user has two factor authentification enabled

locale: The discord language of the user

verified: If the user email is verified

email: The email of the user

flags: The user flags (see: https://discord.com/developers/docs/resources/user#user-object-user-flags)

premium_type:

The nitro level of the user

- 0 : No nitro
- 1 : Basic nitro
- 2 : Nitro boost

public_flags: The public flags of the user (see : https://discord.com/developers/docs/resources/user#user-object-user-flags)

mention: The mention of the user

dm: ***Channel*** The DM channel of the user

2.6 Member

class piscord.**Member**(*member*, *bot*)

Member object is a supplement of the *User* class for guild members

A member is associated with a guild

It have most often the same basic attribute of *User*

nick: The username of member in the guild

roles: ***Role*** List of member roles in the guild if the user if the guild is in the bot cache

roles_id: List of the roles id

hoisted_role: Not implemented

joined_at: Timestamp when member join the guild

premium_since: Timestamp when member starting boost the server

deaf: If the member is deafened in voice channels

mute: If the member is muted in voice channels

guild_id: ID of the guild of the member

guild: ***Guild*** The guild where the member is

add_role(*role*)

Add a role to the guild member

role: ***Role*** A guild role object

ban(*reason=None*)

Ban the guild member

edit(***modifs*)

Modify member, with parameters. Parameters : https://discord.com/developers/docs/resources/guild#modify-guild-member

kick()

Kick the guild member

remove_role(*role*)

Remove a role to the guild member

role: ***Role*** A guild role object

2.7 Reaction

class piscord.**Reaction**(*reaction*, *message*)

Represent a reaction on a message

count: The number of times this emoji was added

me: If the user reacted with this emoji

emoji: ***Emoji*** The emoji of the reaction

message: The message of the reaction

2.8 Emoji

class piscord.**Emoji**(*emoji*)
Represent a Emoji

id: ID of the Emoji

name: The name of the emoji

roles: ***Role*** Roles this emoji is whitelisted to

user: ***User*** User that created this emoji

require_colons: If the emoji must be wrapped in colons

managed: If emoji is managed

animated: If the emoji is animated

available: If the emoji can be used

2.9 Role

class piscord.**Role**(*role*, *bot*)
Represent a Emoji

id: ID of the Emoji

name: The name of the emoji

color: Decimal value of the color

hoist: If the role is pinned in the user listing

position: Position of role in roles list

permissions: Value of role permissions

managed: If role is managed

mentionable: If the role can be mentionned

guild_id: The id of the role guild

mention: The mention of the role

guild: The role guild

delete()
Remove the role

edit(***modifs*)
Modify role, with parameters. Parameters : https://discord.com/developers/docs/resources/guild#modify-guild-role

2.10 Attachment

class piscord.**Attachment**(*attachment={}*)
Represent a message attachment, contain file

id: ID of the attachment

filename: The name of the attachment file

size: The size of the attachment, in bytes

url: Url for get the file

proxy_url: Same than url, but proxied

height: Height of the image attachment (if this is an image)

width: Width of the image attachment (if this is an image)

2.11 Allowed_Mentions

class piscord.**Allowed_Mentions**(*mentions={}*)

Represent the mentions allowed for message sending

parse:

List of the mentions types allowed

- "roles" : mentions of roles
- "users" : mentions of users
- "everyone" : mentions @everyone and @here

roles: List of id of whitelist roles mentions

users: List of id of whitelist users mentions

2.12 Embed

class piscord.**Embed**(*embed={}*)

Represent a message Embed

Params when you send a Embed: https://discord.com/developers/docs/resources/channel#create-message-params

title: The title of the embed

Max : 256 characters

type: Type of embed

Embed types should be considered deprecated and might be removed

- "rich" : Generic Embed
- "image" : Image Embed
- "video" : Video Embed
- "gifv" : Animated gif image Embed rendered as a video Embed
- "article" : Article Embed
- "link" : Link Embed

description: The description of the embed

Max : 2048 characters

url: Url of the Embed

timestamp: Timestamp of Embed content

color: Decimal value of the Embed color

footer: ***Embed_Footer*** The footer of the Embed

image: ***Embed_Image*** The image of the Embed

thumbnail: ***Embed_Image*** The thumbnail of the Embed

video: ***Embed_Image*** The video of the Embed

provider: ***Embed_Provider*** The provider of the Embed

author: ***Embed_Author*** The author informations of the Embed

fields: ***Embed_Field*** List of Embed fields

Max : 25

add_field(***kwargs*)
Used to add a field on the Embed

Return the field

2.12.1 Embed_Field

class piscord.**Embed_Field**(*field*)
Represent a Embed Field

name: The title of the field

Max : 256 characters

value: The description of the field

Max : 1024 characters

inline: If the Embed is inline

2.12.2 Embed_Footer

class piscord.**Embed_Footer**(*footer*)
Represent the footer of an Embed

text: The text of the footer

icon_url: The url of the footer icon

proxy_icon_url: Same than icon_url, but proxied

2.12.3 Embed_Image

class piscord.**Embed_Image**(*image*)
Represent a Embed Image, Embed Thumbnail and Embed Video

url: The url of the image/video

proxy_url: Same than url, but proxied. Video Dmbed does not have this attribute

height: The height of the image

width: The width of the image

2.12.4 Embed_Provider

class piscord.**Embed_Provider**(*provider*)
Represent a Embed Provider

name: The provider name

url: The provider url

2.12.5 Embed_Author

class piscord.**Embed_Author**(*author*)
Represent a Embed Author

name: Name of the author field

Max : 256 characters

url: Url of the author field

icon_url: Url of the icon of the author field

proxy_icon_url: same than icon_url, but proxied

2.13 Invite

class piscord.**Invite**(*invite*, *bot*)
Represent a guild invite

code: The code of the invite (For example, code U9X7XzP corresponding to invitation https://discord.gg/U9X7XzP)

guild: ***Guild*** A partial guild object : The guild of the invite

channel: ***Channel*** A partial channel object : The channel of the invite

inviter: ***User*** The user who created the invite

target_user: ***User*** The target user for the invite

target_user_type:

The type of user target for the invite 1 : Stream

approximate_presence_count: The approximate number of connected users in the guild

approximate_member_count: The approximate number of users in the guild

url: The url of the invite

uses: The number of invite uses

max_uses: The number of max invite uses

max_age: The time before invite is automatically deleted

temporary: If the invite give temporary membership

created_at: Timestamp when the invite was created

delete()
Delete the invite

2.14 Ban

class piscord.**Ban**(*ban*, *bot*)
Represent a guild ban

reason: The reason of the ban

user: ***User*** The banned user

pardon(*guild_id*)
Unban the user

2.15 Overwrite

class piscord.**Overwrite**(*overwrite*, *bot*, *channel_id*)
Represent a Overwrite

id: ID of the Overwrite

type:

The type of the Overwite

- "role" : A role Overwrite
- "member" : A member Overwrite

allow: Permissions allow by Overwrite

deny: Permissions deny by Overwrite

delete()
Delete overwrite

edit(***modifs*)
Modify overwrite, with parameters. Parameters : https://discord.com/developers/docs/resources/channel#edit-channel-permissions

2.16 Webhook

class piscord.**Webhook**(*webhook*, *bot*, *channel=None*)
Represent a channel Webhook

id: ID of the webhook

type:

The type of the webhook

- 0 : Incoming webhook
- 1 : Channel following webhook

guild_id: The id of the webhook guild

channel_id: The id of the webhook channel

user: ***User*** The user who created the webhook

name: The name of the webhook

avatar: The avatar link of the webhook

token: The webhook secure token (For Incoming Webhooks)

channel: ***Channel*** The channel of the webhook

guild: The guild of the webhook

delete()
Delete the webhook

edit(***modifs*)
Modify webhook, with parameters. Parameters : https://discord.com/developers/docs/resources/webhook#modify-webhook

send(*content=None*, *files=None*, ***kwargs*)
Send a message with webhook :

This is like message sending in channel

CHAPTER 3

Permissions

The permissions of any thing in piscord

3.1 List of permissions

CREATE_INSTANT_INVITE: Permission to create instant invites for guild

KICK_MEMBERS: Permission to kick members from the guild

BAN_MEMBERS: Permission to ban members from the guild

ADMINISTRATOR: Admin permission of the guild

MANAGE_CHANNELS: Permission to modify channels of the guild

MANAGE_GUILD: Permission to modify guild

ADD_REACTIONS: Permission to add reaction on a message (if the reaction isn't already here)

VIEW_AUDIT_LOG: Permission to see the guild log

PRIORITY_SPEAKER: Permission to speak on a priority basis in guild

STREAM: Permission to stream in guild voice channel

VIEW_CHANNEL: Permission to view channels in guild

SEND_MESSAGES: Permission to send messages in channel

SEND_TTS_MESSAGES: Permission to send tts messages in channel

MANAGE_MESSAGES: Permission to delete messages of other members

EMBED_LINKS: Permission to auto-embed links

ATTACH_FILES: Permission to send files (images, files..)

READ_MESSAGE_HISTORY: Permission to read previous messages

MENTION_EVERYONE: Permission to mention @everyone or @here

USE_EXTERNAL_EMOJIS: Permission to send emoji from other guilds

VIEW_GUILD_INSIGHTS: Permission to view guild insights

CONNECT: Permission to connect to voice channels

SPEAK: Permission to speak in voice channels

MUTE_MEMBERS: Permission to mute members (can't speak)

DEAFEN_MEMBERS: Permission to deaf members (can't hear)

MOVE_MEMBERS: Permission to move members between voice channels

USE_VAD: Permission to use voice-activity-detection

CHANGE_NICKNAME: Permission to change self nickname

MANAGE_NICKNAMES: Permission to change nickname of other members

MANAGE_ROLES: Permission to create/edit roles

MANAGE_WEBHOOKS: Permission to create/edit webhooks

MANAGE_EMOJIS: Permission to create/edit emojis

3.2 Use Permission

You can use permission to do operations.

1. You can verify permission

```
if role.permissions == Permission.ADMINISTRATOR:
        ...
```

2. You can add a permission

```
role.permissions += Permission.CHANGE_NICKNAME
role.edit(permissions = role.permissions)

# ---------------

perm = channel.permission_overwrites
perm.edit(deny = perm.deny + Permission.SEND_MESSAGES)
```

3. You can remove a permission

```
role.permissions -= Permission.CHANGE_NICKNAME
role.edit(permissions = role.permissions)

# ---------------

perm = channel.permission_overwrites
perm.edit(deny = perm.deny - Permission.SEND_MESSAGES)
```

CHAPTER 4

OAuth with Piscord

Piscord include some functions to use authentification with discord

OAuth is a class to use that

4.1 How to create

```
from piscord import Bot, OAuth

bot = Bot("Token")

auth = OAuth(bot, "Secret", "url of redirect to get token", "scope")
```

You **should** have a bot object but you doesn't need to start it. If you doesn't start the bot, you can create channel, rename, a lot of thing, but can't receive commands or send message (require connexion).

The url is where the user is redirect after authentification with a code get request arg to obtain the token.

The scope is what the bot can do to the user account (see more : https://discord.com/developers/docs/topics/oauth2#shared-resources-oauth2-scopes)

4.2 Get the token

```
# To get the url where redirect the user for authentification with discord
url = auth.get_url()

# Code is the code get in the url after authentification
token = auth.get_token(code)
```

After this, you have the token of the client. You can store it (in user session cookie in web, or anything), and exploit it.

4.3 Usages

```
# Get piscord user objet of the authentified user
# You must have the identify scope
user = auth.get_user(token)

# Get list of the guilds where the user is
# You must have the guilds scope
guilds = auth.get_guilds(token)

# Add the user to a guild where the bot is
# You must have the guilds.join scope
auth.add_guild_member(token, guild_id, user_id)
```

4.4 Functions

class piscord.**OAuth**(*bot*, *secret*, *redirect_uri*, *scope*)

add_guild_member(*token*, *guild_id*, *user_id*)
Add the authentified user to a guild where the bot is

token: The token of the user

guild_id: The if of the guild to add

user_id: The id of the user

get_guilds(*token*)
Get a list of *Guild* objects, the guilds where the user is

token: The token of the user

get_token(*code*)
Get the token of the user

code: The code returned by the authentification

get_url()
Get the url for authentification with discord

get_user(*token*)
Get a *User* object, represent the authentified user

token: The token of the user

Python Module Index

p

Index

O

P

R

S

T

U

W