IreneAPIWrapper

MujyKun

Feb 02, 2023

CONTENTS:

CHAPTER

ONE

CLIENT

class IreneAPIWrapper.models.**IreneAPIClient**(*token: str*, *user_id: Union[int, str]*, *api_url='localhost'*, *port=5454*, *preload_cache: Optional[Preload] = None*, *test=False*, *reconnect=True*, *verbose=False*, *origin='localhost'*, *logger: Optional[Logger] = None*)

Asynchronous IreneAPI Client connected by a websocket.

Warning: It is suggested to only create ONE client per application. The wrapper externally references the latest client created. If it is several clients that are routing to the same API, then it is okay as the same requests will be sent; otherwise, it can lead to conflicts between databases.

Parameters

- **token** (*str*) –
- **user_id** (*Union[int, str]*) –
- **api_url** (*str*) – Defaults to localhost. Websocket URL is expected to be ws://{api_url}:{port}/ws.
- **port** (*int*) – The api port.
- **test** (*bool*) – Whether to go into test/dev mode. Does not currently have a significant difference.
- **reconnect** (*bool*) – Whether to reconnect to the API if a connection is severed.

token

The API token provided.

Type
str

user_id

The id of the user that has access to that token.

Type
str

connected

If there is a stable websocket connection to the API.

Type
bool

`in_testing`

Whether the Client is in testing mode.

Type
bool

`reconnect`

Whether to reconnect to the API if a connection is severed. True by default.

Type
bool

`verbose`

Whether to print verbose messages.

Type
bool

`origin`

The origin meant for CORS to not return a bad request.

Type
str

`logger`

A logging object for messages to be sent to.

Type
logging.Logger

`async add_and_wait(`*callback:* CallBack`)`

Add a callback to the queue and wait for it to complete.

Parameters
callback – The callback to add to the queue and wait for.

`async add_to_queue(`*callback:* CallBack`)`

Add a request to the queue.

Parameters
callback – *CallBack* The request to send to the server.

`async connect()`

Connect to the API via a websocket indefinitely.

`async disconnect()`

Disconnect from the current websocket connection.

`property is_preloaded`

Check if the client is preloaded with cache.

CHAPTER

TWO

ABSTRACT BASE CLASSES

2.1 AbstractModel

class IreneAPIWrapper.models.**AbstractModel**(*obj_id*)

async static create(**args*, ***kwargs*)

Create an object.

async static create_bulk(*list_of_dicts: List[dict]*)

Bulk create objects

Parameters
list_of_dicts – List[dict] A list of dictionaries.

async delete()

Delete the current object from the database and remove it from cache.

async static fetch(*unique_id: int*)

Fetch the object from the API.

async static fetch_all()

Fetch all objects from the API.

async static get(*unique_id: int*, *fetch: bool*)

Get an object if it exists in cache, otherwise fetch the object from the API.

async get_card(*markdown=False*)

Get a list representing of the current object as a card.

Parameters
markdown – bool Whether the returned list should support markdown.

Returns
List[str] A list of strings for the card.

async static insert(**args*, ***kwargs*)

Insert a new object into the database.

2.2 MediaSource

class IreneAPIWrapper.models.**MediaSource**(*url*, *media_id: Optional[int] = None*, *file_type=None*)

Represents a MediaSource object.

A MediaSource object inherits from *File*.

media_id

The Media id.

Type
int

file_type

The file type (if it is known).

Type
str

url

The URL of the media.

Type
str

async **download_and_get_image_host_url**() → str

Download and get the image host url if possible, otherwise fallback to the default url.

Returns
str A image host url or fallbacks to the default url.

2.3 Alias

class IreneAPIWrapper.models.**Alias**(*alias_id*, *alias_name*, *obj_id*, *guild_id*)

Represents an Abstract Alias.

An Alias object inherits from *AbstractModel*.

Please note that concrete aliases of different types will overlap unique keys, so they must have their own cache per concrete type of alias.

Parameters

- **alias_id** (*int*) – The Alias id.
- **alias_name** (*str*) – The alias name.
- **obj_id** (*int*) – The ID of the object the alias is referring to.
- **guild_id** (*Optional[int]*) – A guild ID that owns the alias if there is one.

id

The Alias id.

Type
int

name

The alias name.

Type
str

_obj_id

The ID of the object the alias is referring to. Used for Abstraction.

Type
int

guild_id

A guild ID that owns the alias if there is one.

Type
Optional[int]

2.4 File

class IreneAPIWrapper.models.**File**(*file_type=None*)

2.5 Receiver

async IreneAPIWrapper.models.base.receiver.**internal_delete**(*obj:* AbstractModel, *request: dict*) → *CallBack*

Delete the known instance of the concrete object from the API.

> **Warning:** This is a permanent deletion from the database. Concrete objects are removed from cache on deletion.

Parameters

- **obj** – *AbstractModel* An abstract model.
- **request** – dict The request to pass into a *CallBack*.

Returns
CallBack

async IreneAPIWrapper.models.base.receiver.**internal_fetch**(*obj:* AbstractModel, *request: dict*) → Optional[*AbstractModel*]

Fetch an updated concrete object from the API.

> **Note:** Concrete objects are added to cache on creation.

Parameters

- **obj** – *AbstractModel* An abstract model.
- **request** – dict The request to pass into a Callback.

Returns
AbstractModel Returns an abstract model.

async IreneAPIWrapper.models.base.receiver.**internal_fetch_all**(*obj:* AbstractModel, *request: dict*, *bulk: bool = False*) → List[*AbstractModel*]

Fetch all known instances of the concrete object from the API.

Note: Concrete objects are added to cache on creation.

Parameters

- **obj** – *AbstractModel* An abstract model.
- **request** – dict The request to pass into a Callback.
- **bulk** – bool Whether to generate objects in bulk (Defaults to False).

Returns
List[*AbstractModel*] Returns a list of abstract models.

async IreneAPIWrapper.models.base.receiver.**internal_insert**(*request: dict*) → *CallBack*

Insert an object into the database.

Parameters
request – dict The request to pass into a Callback.

Returns
CallBack Returns a *CallBack* object.

CHAPTER

THREE

API MODELS

3.1 Channel

class IreneAPIWrapper.models.**Channel**(*channel_id*, *guild_id=None*)

Represents a discord channel.

A Channel object inherits from *AbstractModel*.

Parameters
channel_id (*int*) – The channel id.

id

The channel id.

Type
int

guild_id

The guild ID.

Type
Optional[int]

async static create(**args*, ***kwargs*)

Create a Channel object.

Returns
Channel

async delete() → None

Delete a Channel object from the database and remove it from cache.

> **Warning:** This will cascade all objects dependent on the object.

Returns
None

async static fetch(*channel_id*)

Fetch an updated channel object from the API.

Parameters
channel_id – int The channel ID to fetch.

Returns
Optional[*Channel*] The channel object requested.

`async static fetch_all()`
Fetch all Channels.

Note: Channel objects are added to cache on creation.

`async static get(channel_id: int, fetch=True)`
Get a Channel object.

If the Channel object does not exist in cache, it will fetch the name from the API.

Parameters
- **channel_id** – int The channel ID to retrieve.
- **fetch** – bool Whether to fetch from the API if not found in cache.

Returns
Optional[*Channel*] The channel object requested.

`async static get_all()`
Get all Channel objects in cache.

Returns
dict_values[*Channel*] All Channel objects from cache.

`async static insert(channel_id, guild_id)` → None
Insert a new channel into the database.

Parameters
- **channel_id** – The channel ID to insert.
- **guild_id** – The guild ID to insert.

Returns
None

3.2 Interaction

`class IreneAPIWrapper.models.Interaction(interaction_type: InteractionType, url: str)`
Represents an Interaction object.

An Interaction object inherits from *AbstractModel*.

Parameters
- **interaction_type** (*InteractionType*) – InteractionType
- **url** (*str*) – Interaction URL.

`id`
InteractionType ID - URL

Type
str

type

InteractionType

Type
InteractionType

url

Interaction URL.

Type
str

async static create(**args*, ***kwargs*)

Create an Interaction object.

Returns
Optional[*Interaction*]

async delete() → None

Delete the Interaction object from the database and remove it from cache.

Returns
None

async static fetch_all()

Fetch all Interactions.

async static get_all()

Get all Interaction objects in cache.

Returns
dict_values[*Interaction*] All Interaction objects from cache.

async static insert(*type_id: int*, *url: str*) → None

Insert a new interaction into the database.

Parameters

- **type_id** – int The Interaction Type.
- **url** – str The interaction url.

Returns
None

3.3 InteractionType

class IreneAPIWrapper.models.**InteractionType**(*type_id*, *name*)

Represents an InteractionType object.

An InteractionType object inherits from *AbstractModel*.

Parameters

- **type_id** (*int*) – Type ID
- **name** (*str*) – Name of the interaction type.

id

Type ID

Type
int

name

Name of the interaction type.

Type
str

async delete() → None

Delete the InteractionType object from the database and remove it from cache.

Returns
None

async static get(*type_id: int*)

Get an InteractionType object.

If the InteractionType object does not exist in cache :param type_id: int

The ID of the InteractionType

Returns
Optional[*InteractionType*]

async static get_all()

Get all Interaction Type objects in cache.

Returns
dict_values[*InteractionType*] All Interaction Type objects from cache.

async static insert(*name: str*) → None

Insert a new interaction type into the database.

Parameters
name – str The type name.

Returns
None

3.4 Notification

class IreneAPIWrapper.models.**Notification**(*noti_id*, *guild_id*, *user_id*, *phrase*)

Represents a Notification.

A Notification object inherits from *AbstractModel*.

Note: One Notification object will be referenced as "noti" and not "notification". Several notifications will be referenced as "notifications" and not "notis"

Parameters

- **noti_id** (*int*) – The noti's ID.

- **guild_id** (*int*) – Guild ID of the noti.
- **user_id** (*int*) – User ID to notify.
- **phrase** (*str*) – The phrase to notify the user for.

id

The noti's ID.

Type
int

guild_id

Guild ID of the noti.

Type
int

user_id

User ID to notify.

Type
int

phrase

The phrase to notify the user for.

Type
str

async static create(**args*, ***kwargs*)

Create a Noti object.

async delete() → None

Delete the Noti object from the database and remove it from cache.

Returns
None

async static fetch(*noti_id: int*)

Fetch an updated noti object from the API.

Parameters
noti_id – int The noti's ID to fetch.

Returns
Notification

async static fetch_all()

Fetch all Notifications.

async static get(*noti_id: int*, *fetch=True*)

Get a Noti object.

If the Noti object does not exist in cache, it will fetch the person from the API. :param noti_id: int

The ID of the Noti to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Notification

async static get_all(*guild_id=None*, *user_id=None*)

Get Notification objects in cache (can be filtered).

Parameters

- **guild_id** – int Guild ID to filter by
- **user_id** – int User ID to filter by

Returns
dict_values[*Notification*] All Notification objects from cache.

async static insert(*guild_id*, *user_id*, *phrase*) → None

Insert a new Noti into the database and cache.

Parameters

- **guild_id** (*int*) – Guild ID.
- **user_id** (*int*) – User ID to be notified.
- **phrase** (*str*) – Phrase to notify the user for.

:param : :type : returns: None

3.5 BiasGame

class IreneAPIWrapper.models.**BiasGame**

async static fetch_winners(*user_id*, *limit=15*) → dict

Fetch the winners of a user's bias game in DESC order.

Parameters

- **user_id** – int User ID to return results for.
- **limit** – int Number of results should be retrieved in descending order.

Returns
dict Dictionary of person IDs to the amount of times they've won.

async static generate_bracket(*game_info*)

Generate a PvP bracket and return an image url.

Parameters
game_info – dict All BiasGame round(s) information.

Returns
str The BiasGame bracket image url.

async static generate_pvp(*first_image_url*, *second_image_url*)

Generate a PvP image and return an image url.

Parameters

- **first_image_url** – str The first image url.
- **second_image_url** – str The second image url.

Returns
str The PvP image url.

async static upsert_win(*user_id*, *person_id*) → None

Upsert a win for a user's BiasGame.

Parameters

- **user_id** – int User ID of the bias game player.
- **person_id** – int Person ID that won the bias game.

Returns
None

3.6 GuessingGame

class IreneAPIWrapper.models.**GuessingGame**(*game_id: int*, *date_id: int*, *media_ids: List[int]*, *status_ids: List[int]*, *mode_id: int*, *difficulty:* Difficulty, *is_nsfw: bool*)

Represents a Guessing Game.

A GuessingGame object inherits from *AbstractModel*.

Parameters

- **date_id** (*int*) – The date object ID.
- **media_ids** (*List[int]*) – The media IDs.
- **status_ids** (*List[int]*) – The status IDs.
- **mode_id** (*int*) – The mode of the game.
- **difficulty** (*Difficulty*) – The difficulty of the game.
- **is_nsfw** (*bool*) – Whether the content may be NSFW.

date_id

The date object ID.

Type
int

media_ids

The media IDs.

Type
List[int]

status_ids

The status IDs.

Type
List[int]

mode_id

The mode of the game.

Type
int

difficulty

The difficulty of the game.

Type
Difficulty

is_nsfw

Whether the content may be NSFW.

Type
bool

async static create(**args*, ***kwargs*)

Create a GuessingGame object.

Returns
GuessingGame

async delete() → None

Delete the GuessingGame object from the database and remove it from cache.

Returns
None

async static fetch(*game_id: int*)

Fetch an updated GuessingGame object from the API.

Note: affiliation objects are added to cache on creation.

Parameters
game_id – int The GuessingGame's ID to fetch.

Returns
Optional[*GuessingGame*] The GuessingGame object requested.

async static fetch_all()

Fetch all GuessingGame objects.

Note: GuessingGame objects are added to cache on creation.

async static get(*game_id: int*, *fetch=True*)

Get a GuessingGame object.

If the GuessingGame object does not exist in cache, it will fetch the id from the API. :param game_id: int

The ID of the GuessingGame to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Optional[*GuessingGame*] The GuessingGame object requested.

async static get_all()

Get all GuessingGame objects in cache.

Returns
dict_values[*GuessingGame*] All GuessingGame objects from cache.

async static insert(*date_id: int*, *media_ids: List[int]*, *status_ids: List[int]*, *mode_id: int*, *difficulty_id: int*, *is_nsfw: bool*) → int

Insert a new GuessingGame into the database.

Parameters

- **date_id** – int The Date ID
- **media_ids** – List[int] A list of media object ids.
- **status_ids** – List[int] A list of status ids
- **mode_id** – int The guessing game's mode.
- **difficulty_id** – int The difficulty of the guessing game.
- **is_nsfw** – bool Whether the game includes nsfw content.

Returns
int The guessing game ID.

async update_media_and_status(*media_ids: List[int]*, *status_ids: List[int]*) → None

Update the media and status ids for the game in the database.

Returns
None

3.7 UnscrambleGame

class IreneAPIWrapper.models.**UnscrambleGame**(*game_id: int*, *date_id: int*, *status_ids: List[int]*, *mode_id: int*, *difficulty:* Difficulty)

Represents an UnscrambleGame Game.

A UnscrambleGame object inherits from *AbstractModel*.

Parameters

- **date_id** (*int*) – The date object ID.
- **status_ids** (*List[int]*) – The status IDs.
- **mode_id** (*int*) – The mode of the game.
- **difficulty** (*Difficulty*) – The difficulty of the game.

date_id

The date object ID.

Type
int

status_ids

The status IDs.

Type
List[int]

`mode_id`

The mode of the game.

Type
int

`difficulty`

The difficulty of the game.

Type
Difficulty

`async static create`(**args*, ***kwargs*)

Create a UnscrambleGame object.

Returns
UnscrambleGame

`async delete`() → None

Delete the UnscrambleGame object from the database and remove it from cache.

Returns
None

`async static fetch`(*game_id: int*)

Fetch an updated UnscrambleGame object from the API.

Note: affiliation objects are added to cache on creation.

Parameters
game_id – int The UnscrambleGame's ID to fetch.

Returns
Optional[*UnscrambleGame*] The UnscrambleGame object requested.

`async static fetch_all`()

Fetch all UnscrambleGame objects.

Note: UnscrambleGame objects are added to cache on creation.

`async static get`(*game_id: int*, *fetch=True*)

Get a UnscrambleGame object.

If the UnscrambleGame object does not exist in cache, it will fetch the id from the API. :param game_id: int

The ID of the UnscrambleGame to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Optional[*UnscrambleGame*] The UnscrambleGame object requested.

async static get_all()

Get all UnscrambleGame objects in cache.

Returns
dict_values[*UnscrambleGame*] All UnscrambleGame objects from cache.

async static insert(*date_id: int*, *status_ids: List[int]*, *mode_id: int*, *difficulty_id: int*) → int

Insert a new UnscrambleGame into the database.

Parameters

- **date_id** – int The Date ID
- **status_ids** – List[int] A list of status ids
- **mode_id** – int The guessing game's mode.
- **difficulty_id** – int The difficulty of the guessing game.
- **is_nsfw** – bool Whether the game includes nsfw content.

Returns
int The guessing game ID.

async update_status(*status_ids: List[int]*) → None

Update the status ids for the game in the database.

Returns
None

3.8 Wolfram

class IreneAPIWrapper.models.**Wolfram**

A model for sending requests to WolframAlpha.

async static query(*query*)

Query a request to Wolfram.

3.9 EightBallResponse

class IreneAPIWrapper.models.**EightBallResponse**(*response_id: int*, *response: str*)

Represents an eight-ball Response.

An EightBallResponse object inherits from *AbstractModel*.

Parameters

- **response_id** (*int*) – The response id.
- **response** (*str*) – The response itself.

id

The response id.

Type
int

response

The response itself.

Type
str

async static create(**args*, ***kwargs*)

Create an EightBallResponse object.

Returns
EightBallResponse

async delete() → None

Delete the Response object from the database and remove it from cache.

Returns
None

async static fetch(*response_id: int*)

Fetch an updated EightBallResponse object from the API.

Note: EightBallResponse objects are added to cache on creation.

Parameters
response_id – int The response's ID to fetch.

Returns
Optional[*EightBallResponse*] The EightBallResponse object requested.

async static fetch_all()

Fetch all responses.

Note: EightBallResponse objects are added to cache on creation.

async static get(*response_id: int*, *fetch=True*)

Get an EightBallResponse object.

If the EightBallResponse object does not exist in cache, it will fetch the object from the API. :param response_id: int

The ID of the Response to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Optional[*EightBallResponse*] The EightBallResponse object requested.

async static get_all()

Get all EightBallResponse objects in cache.

Returns
dict_values[*EightBallResponse*] All EightBallResponse objects from cache.

async static get_random_response(*fetch=False*)

Get a random response

Parameters
fetch – bool Whether to fetch fresh results from the API.

Returns
EightBallResponse A random response object.

async static insert(*response: str*) → None

Insert a new EightBallResponse into the database and cache.

Parameters
response – str Response Message.

3.10 PackMessage

class IreneAPIWrapper.models.**PackMessage**(*language_id*, *label*, *message*, *num_inputs*)

Refers to a message in a language.

Parameters

- **language_id** (*int*) – The *Language* ID that the message belongs to.
- **label** (*str*) – The key that identifies the message.
- **message** (*str*) – The message that may have custom input.
- **num_inputs** (*int*) – The number of custom inputs.

language_id

The *Language* ID that the message belongs to.

Type
int

label

The key that identifies the message.

Type
str

message

The message that may have custom input.

Type
str

num_inputs

The number of custom inputs.

Type
int

get(**args*) → str

Get the message formatted with custom input.

Pass in as many strings as inputs are required.

async static get_input_count(*msg: str*) → int

Get the amount of inputs in a message.

Parameters
msg – str The message to check.

Returns
int The number of inputs in the input message.

3.11 Language

class IreneAPIWrapper.models.**Language**(*language_id*, *short_name*, *name*, *pack: List[*PackMessage*]*)

Refers to a language.

A Language object inherits from *AbstractModel*.

Parameters

- **language_id** (*int*) – The language's unique ID.
- **short_name** (*str*) – The shorthand version of the language's name.
- **name** (*str*) – The official name of the language.
- **pack** (List[*PackMessage*]) – A list of PackMessages that belong to the language.

short_name

async static create(**args*, ***kwargs*)

Create an object.

async static fetch_all()

Fetch all objects from the API.

static get_english()

Get the English Language Pack.

Returns
Language

static get_lang(*short_name: str*)

Get a language by the short name.

Parameters
short_name – str The short name of a language.

Returns
Optional[*Language*] The language object.

static get_lang_by_id(*language_id*)

Get a language by the ID.

Parameters
language_id – int The ID of the language.

Returns
Optional[*Language*] The language object.

3.12 Guild

class IreneAPIWrapper.models.**Guild**(*guild_id*, *name*, *emoji_count*, *afk_timeout*, *icon*, *owner_id*, *owner*, *banner*, *description*, *mfa_level*, *splash*, *nitro_level*, *boosts*, *text_channel_count*, *voice_channel_count*, *category_count*, *emoji_limit*, *member_count*, *role_count*, *shard_id*, *create_date*, *has_bot*, *prefixes=None*)

Represents a Guild object.

A Guild inherits from *AbstractModel*.

Parameters

- **guild_id** (*int*) – The guild's id.
- **name** (*str*) – The guild name.
- **emoji_count** (*int*) – Current amount of emojis on the guild.
- **afk_timeout** (*int*) – The time before a User gets timed out in a voice channel.
- **icon** (*str*) – The Guild's icon.
- **owner_id** (*int*) – The guild owner's id.
- **owner** (*User*) – A reference to the guild owner's User object.
- **banner** (*str*) – A banner of the guild.
- **description** (*str*) – A description of the guild.
- **mfa_level** (*int*) – MFA level of the guild.
- **splash** (*str*) – Splash Art of the guild.
- **nitro_level** (*int*) – Current nitro level of the guild.
- **boosts** (*int*) – Number of times the guild has been boosted.
- **text_channel_count** (*int*) – Amount of text channels the guild has.
- **voice_channel_count** (*int*) – Amount of voice channels the guild has.
- **category_count** (*int*) – Amount of categories the guild has.
- **emoji_limit** (*int*) – Maximum amount of emojis the guild can have.
- **member_count** (*int*) – Amount of users the guild has.
- **role_count** (*int*) – Amount of roles the guild has.
- **shard_id** (*int*) – The shard connected to the guild.
- **create_date** (*str*) – The date the guild was created.
- **has_bot** (*bool*) – Whether the bot exists in the guild.
- **prefixes** (*Optional[List[str]]*) – A list of prefixes the guild uses.

id

The guild's id.

Type
int

`name`

The guild name.

Type
str

`emoji_count`

Current amount of emojis on the guild.

Type
int

`afk_timeout`

The time before a User gets timed out in a voice channel.

Type
int

`icon`

The Guild's icon.

Type
str

`owner_id`

The guild owner's id.

Type
int

`owner`

A reference to the guild owner's User object.

Type
User

`banner`

A banner of the guild.

Type
str

`description`

A description of the guild.

Type
str

`mfa_level`

MFA level of the guild.

Type
int

`splash`

Splash Art of the guild.

Type
str

`nitro_level`

Current nitro level of the guild.

Type
int

`boosts`

Number of times the guild has been boosted.

Type
int

`text_channel_count`

Amount of text channels the guild has.

Type
int

`voice_channel_count`

Amount of voice channels the guild has.

Type
int

`category_count`

Amount of categories the guild has.

Type
int

`emoji_limit`

Maximum amount of emojis the guild can have.

Type
int

`member_count`

Amount of users the guild has.

Type
int

`role_count`

Amount of roles the guild has.

Type
int

`shard_id`

The shard connected to the guild.

Type
int

`create_date`

The date the guild was created.

Type
str

has_bot

Whether the bot exists in the guild.

Type
bool

prefixes

A list of prefixes the guild uses.

Type
Optional[List[str]]

async add_prefix(*prefix: str*) → None

Add a guild prefix.

Parameters
prefix – str The prefix to add.

async static create(**args*, ***kwargs*)

Create a Guild object.

Returns
Guild

async delete() → None

Delete the Guild object from the database and remove it from cache.

Returns
None

async delete_prefix(*prefix: str*) → None

Delete a guild prefix.

Parameters
prefix – str The prefix to delete.

async static fetch(*guild_id: int*)

Fetch an updated Guild object from the API.

Parameters
guild_id – int The guild's ID to fetch.

Returns
Guild

async static fetch_all()

Fetch all Guild objects from the API.

Returns
List[*Guild*]

async static fetch_all_prefixes() → Dict[int, List[str]]

Fetch all prefixes.

Returns
Dict[int, List[str]]

async fetch_prefixes() → List[str]

Get a list of prefixes for the guild from the API.

Returns
List[str]

async static get(*guild_id: int*, *fetch=True*)

Get a Guild object.

If the Guild object does not exist in cache, it will fetch the name from the API. :param guild_id: int

The ID of the guild to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Guild

async static get_all()

Get all Guild objects in cache.

Returns
dict_values[*Guild*] All Guild objects from cache.

async static insert(*guild_id*, *name=None*, *emoji_count=None*, *afk_timeout=None*, *icon=None*, *owner_id=None*, *banner=None*, *description=None*, *mfa_level=None*, *splash=None*, *nitro_level=None*, *boosts=None*, *text_channel_count=None*, *voice_channel_count=None*, *category_count=None*, *emoji_limit=None*, *member_count=None*, *role_count=None*, *shard_id=None*, *create_date=None*, *has_bot=True*)

Insert a new Guild into the database.

Parameters

- **guild_id** (*int*) – The guild's ID.
- **name** (*str*) – Name of the guild.
- **emoji_count** (*int*) – Number of emojis the guild has.
- **afk_timeout** (*int*) – Guild AFK timeout for voice channels
- **icon** (*str*) – Icon URL of the guild.
- **owner_id** (*int*) – Owner ID of the guild.
- **banner** (*str*) – Banner of the guild.
- **description** (*str*) – Guild description.
- **mfa_level** (*int*) – MFA level of the guild.
- **splash** (*str*) – Splash art url of the guild.
- **nitro_level** (*int*) – Nitro level of the guild.
- **boosts** (*int*) – Number of boosts the guild has.
- **text_channel_count** (*int*) – Number of text channels the guild has.
- **voice_channel_count** (*int*) – Number of voice channels the guild has.
- **category_count** (*int*) – Number of categories the guild has.
- **emoji_limit** (*int*) – Maximum number of emojis allowed in the guild.
- **member_count** (*int*) – The number of members in the guild.
- **role_count** (*int*) – The number of roles in the guild.
- **shard_id** (*int*) – The shard the guild is connected to.

- **create_date** (*timestamp*) – The creation date of the Guild.
- **has_bot** (*bool*) – Whether the guild has the bot.

:param : :type : returns: None

3.13 Media

class IreneAPIWrapper.models.**Media**(*media_id*, *source*, *faces*, *affiliation*, *is_enabled*, *is_nsfw*, *failed=0*, *correct=0*)

Represents a Media object.

A Media object inherits from *AbstractModel*.

Parameters

- **media_id** (*int*) – The Media ID.
- **source** (*MediaSource*) – The *MediaSource* object that contains information about the media source.
- **faces** (*int*) – The amount of faces detected in the image.
- **affiliation** (*Affiliation*) – The *Affiliation* associated with the media.
- **is_enabled** (*bool*) – If the media is enabled for usage.
- **is_nsfw** (*bool*) – If the media may contain explicit content.

id

The Media ID.

Type
int

source

The *MediaSource* object that contains information about the media source.

Type
MediaSource

faces

The amount of faces detected in the image.

Type
int

affiliation

The *Affiliation* associated with the media.

Type
Affiliation

is_enabled

If the media is enabled for usage.

Type
bool

is_nsfw

If the media may contain explicit content.

Type
bool

async static create(**args*, ***kwargs*)

Create a Media object.

Returns
Media

async delete() → None

Delete the Media object from the database and remove it from cache.

Returns
None

property difficulty

Get the difficulty (ratio) of the media.

async static fetch(*object_id: int*, *affiliation=False*, *person=False*, *group=False*)

Fetch an updated Media object from the API.

Parameters

- **object_id** – int The object ID to fetch. This can be an affiliation, person, group, or media (default) ID if specified.
- **affiliation** – bool If the object ID is an Affiliation ID.
- **person** – bool If the object ID is a Person ID.
- **group** – bool If the object ID is a Group ID.

async static fetch_all()

Fetch all media.

async static get(*media_id: int*, *fetch=True*)

Get a media object.

If the Media object does not exist in cache, it will fetch the name from the API. :param media_id: int

The ID of the media to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

async static get_all(*affiliations: Optional[List[*Affiliation*]] = None*, *limit=None*, *count_only=False*)

Get all Media objects in cache or from the API.

Parameters

- **affiliations** – Optional[List[*Affiliation*]] A list of affiliations that must belong with the media.
- **limit** – Optional[int] A maximum number of results to be sent if fetched.
- **count_only** – bool Whether to only return the number of available media.

Returns
dict_values[*Media*] All Media objects from cache/API.

Returns
int The number of media objects available for the affiliations if count_only is set to True.

`async static get_random`(*object_id: int*, *affiliation=False*, *person=False*, *group=False*, *min_faces=1*, *max_faces=999*, *can_be_nsfw=False*, *is_enabled=True*, *file_type=None*)

Get a random Media object from the API.

Parameters

- **object_id** – int The object ID to grab media for. This can be an affiliation, person, group, or media (default) ID if specified.
- **affiliation** – bool If the object ID is an Affiliation ID.
- **person** – bool If the object ID is a Person ID.
- **group** – bool If the object ID is a Group ID.
- **min_faces** – int Minimum number of faces the media can have
- **max_faces** – int Maximum number of faces the media can have
- **can_be_nsfw** – bool If it is okay to have NSFW media.
- **is_enabled** – bool The media object is officially active.
- **file_type** – str A restricted file type.

`async static insert`(*link*, *face_count*, *file_type*, *affiliation_id*, *enabled*, *is_nsfw*) → None

Insert a new media into the database.

Parameters

- **link** – str The link of the media.
- **face_count** – int Number of faces in the media.
- **file_type** – str The file type of the media.
- **affiliation_id** – int The affiliation ID associated with the media.
- **enabled** – bool Whether the media will be active for searches.
- **is_nsfw** – Whether the media may be NSFW.

Returns
None

`async upsert_guesses`(*correct: bool*)

Increment the guesses appropriately and update to the database after the total amount is divisible by 5.

Parameters
correct – bool Whether the user guessed correctly.

`property url`

Get the media source url.

3.14 User

class IreneAPIWrapper.models.**User**(*user_id*, *is_patron*, *is_super_patron*, *is_banned*, *is_mod*, *is_data_mod*, *is_translator*, *is_proofreader*, *balance*, *xp*, *api_access*, *gg_filter_active*, *language*, *lastfm*, *timezone*, *rob_level*, *daily_level*, *beg_level*, *profile_level*, *gg_filter_person_ids*, *gg_filter_group_ids*)

async add_token(*unhashed_token*, *access:* Access)

Add an API token to a user.

Parameters

- **unhashed_token** –
- **access** – (Access) An Access object that is predefined in models/access

async static create(**args*, ***kwargs*)

Create a User object.

Returns
User

async delete() → None

Delete a user from the database and wipe all the information about them.

Returns
None

async delete_token()

Delete the user's current API token if they have one.

async static fetch(*user_id: int*)

Fetch an updated User object from the API.

If the user is not in the DB, it will add it. .. NOTE:: User objects are added to cache on creation.

Parameters
user_id – int The user's ID to fetch.

Returns
User

async static fetch_all()

Fetch all users.

Note: User objects are added to cache on creation.

---

async static get(*user_id: int*, *fetch=True*)

Get a User object.

If the User object does not exist in cache, it will fetch the user from the API. :param user_id: int

The ID of the user to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
User

async static get_all()

Get all User objects in cache.

Returns
dict_values[*User*] All User objects from cache.

async static insert(*user_id: int*)

Add a user to the database.

Parameters
user_id – The user ID to add.

Returns
None

async set_ban(*active=True*)

Ban or Unban the user from the bot.

Parameters
active – (bool) Whether the ban should be active.

async set_data_mod(*active=True*)

Add or Revoke a user's data moderator status.

Parameters
active – Whether the status should be active.

async set_mod(*active=True*)

Add or Revoke a user's moderator status.

Parameters
active – Whether the status should be active.

async set_patron(*active=True*)

Add or Revoke a user's patron status.

Parameters
active – Whether the status should be active.

async set_proofreader(*active=True*)

Add or Revoke a user's proofreader status.

Parameters
active – Whether the status should be active.

async set_super_patron(*active=True*)

Add or Revoke a user's super patron status.

Parameters
active – Whether the status should be active.

async set_translator(*active=True*)

Add or Revoke a user's translator status.

Parameters
active – Whether the status should be active.

async upsert_filter_groups(*group_ids: Tuple[int]*)

Upsert groups to the gg filter.

Parameters
group_ids – Tuple[int] A tuple of group ids that the user should have.

async upsert_filter_persons(*person_ids: Tuple[int]*)

Upsert persons to the gg filter.

Parameters
person_ids – Tuple[int] A tuple of person ids that the user should have.

3.15 Affiliation

class IreneAPIWrapper.models.**Affiliation**(*affiliation_id: int*, *person:* Person, *group:* Group, *positions: Optional[List[*Position*]]*, *stage_name: str*)

Represents the connection between a Person and Group object.

An Affiliation object inherits from *AbstractModel*.

Parameters

- **affiliation_id** (*int*) – The Affiliation id.
- **person** (*Person*) – The person that is affiliated with a Group.
- **group** (*Group*) – The group that the Person is affiliated with.
- **positions** (Optional[List[*Position*]]) – The positions that the Person has in the Group.
- **stage_name** (*str*) – Exclusive name of the Person when they are in the Group.

id

The Affiliation id.

Type
int

person

The person that is affiliated with a Group.

Type
Person

group

The group that the Person is affiliated with.

Type
Group

positions

The positions that the Person has in the Group.

Type
List[*Position*]

stage_name

Exclusive name of the Person when they are in the Group.

Type
str

async static create(**args*, ***kwargs*)

Create an Affiliation object.

Returns
Affiliation

`async delete()` → None

Delete the Affiliation object from the database and remove it from cache.

Returns
None

`async static fetch(affiliation_id: int)`

Fetch an updated affiliation object from the API.

Note: affiliation objects are added to cache on creation.

Parameters
affiliation_id – int The affiliation's ID to fetch.

Returns
Optional[*Affiliation*] The affiliation object requested.

`async static fetch_all()`

Fetch all affiliations.

Note: affiliation objects are added to cache on creation.

`async static get(affiliation_id: int, fetch=True)`

Get an Affiliation object.

If the Affiliation object does not exist in cache, it will fetch the name from the API. :param affiliation_id: int

The ID of the affiliation to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Optional[*Affiliation*] The affiliation object requested.

`async static get_all()`

Get all Affiliation objects in cache.

Returns
dict_values[*Affiliation*] All Affiliation objects from cache.

`async get_card(markdown=False, extra=True)`

Get a list representing of the current object as a card.

Parameters
markdown – bool Whether the returned list should support markdown.

Returns
List[str] A list of strings for the card.

async static insert(*person_id: int*, *group_id: int*, *position_ids: List[int]*, *stage_name: str*) → bool

Insert a new affiliation into the database.

Parameters

- **person_id** – int The person id to affiliate with a Group.
- **group_id** – int The group id that is affiliated with a Person.
- **position_ids** – List[int] The Positions the Person has in the Group.
- **stage_name** – str The exclusive name of the Person in the Group.

Returns

bool Whether the affiliation was added to the existing objects as well as inserted into the DB.

3.16 Person

class IreneAPIWrapper.models.**Person**(*person_id*, *date*, *name*, *former_name*, *display*, *social*, *location*, *blood_type*, *gender*, *description*, *height*, *call_count*, *media_count*, *tags*, *aliases*)

Represents a Person (or a living entity).

A Person object inherits from *AbstractModel*.

Note: Several Person objects will be referenced as "persons" and not "people".

Parameters

- **person_id** (*int*) – The person's ID.
- **date** (*Date*) – Birth/Death date of a person.
- **name** (*Name*) – The official Name of the person.
- **former_name** (*Name*) – The former Name object of the person.
- **display** (*Display*) – The avatar/banner displays for the person.
- **social** (*Social*) – All social media references for the person.
- **location** (*Location*) – The birth location of the person.
- **blood_type** (*BloodType*) – The blood type of the person.
- **gender** (*str*) – The gender of the person.
- **description** (*str*) – A general overview of the person.
- **height** (*int*) – The height of the person in centimeters (cm)
- **call_count** (*int*) – The amount of times the person has been called. (Increment determined by client side and not from the API)
- **media_count** (*int*) – The media a person has.
- **tags** (List[*Tag*]) – The tags associated with the person.
- **aliases** (List[*PersonAlias*]) – The aliases associated with the person.

id

The person's ID.

Type
int

`date`
Birth/Death date of a person.

Type
Date

`name`
The official Name of the person.

Type
Name

`former_name`
The former Name object of the person.

Type
Name

`display`
The avatar/banner displays for the person.

Type
Display

`social`
All social media references for the person.

Type
Social

`location`
The birth location of the person.

Type
Location

`blood_type`
The blood type of the person.

Type
BloodType

`gender`
The gender of the person.

Type
str

`description`
A general overview of the person.

Type
str

`height`
The height of the person in centimeters (cm)

Type
int

call_count

The amount of times the person has been called. (Increment determined by client side and not from the API)

Type
int

media_count

The media a person has.

Type
int

tags

The tags associated with the person.

Type
List[*Tag*]

aliases

The aliases associated with the person.

Type
List[*PersonAlias*]

affiliations

A list of *Affiliation* objects between the *Person* and the *Group* objects they are in.

Type
List[*Affiliation*]

async static create(**args*, ***kwargs*)

Create a Person object.

async delete() → None

Delete the Person object from the database and remove it from cache.

Returns
None

async static fetch(*person_id: int*)

Fetch an updated Person object from the API.

Parameters
person_id – int The person's ID to fetch.

Returns
Person

async static fetch_all()

Fetch all persons.

async static get(*person_id: int*, *fetch=True*)

Get a Person object.

If the Person object does not exist in cache, it will fetch the person from the API. :param person_id: int

The ID of the person to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Person

async static get_all()

Get all Person objects in cache.

Returns
dict_values[*Person*] All Person objects from cache.

async get_card(*markdown=False*, *extra=True*)

Get a list representing of the current object as a card.

Parameters
markdown – bool Whether the returned list should support markdown.

Returns
List[str] A list of strings for the card.

async static insert(*date_id*, *name_id*, *former_name_id*, *gender*, *description*, *height*, *display_id*, *social_id*, *location_id*, *tag_ids*, *blood_id*, *call_count*) → None

Insert a new person into the database.

Parameters

- **date_id** (*int*) – The *Date* ID of the person.
- **name_id** (*int*) – The official *Name* ID of the person.
- **former_name_id** (*int*) – The former *Name* ID of the person.
- **gender** (*str*) – The gender of the person.
- **description** (*str*) – An overview of the person.
- **height** (*int*) – The height of the person in centimeters (cm).
- **display_id** (*int*) – The *Display* ID of the person.
- **social_id** (*int*) – The *Social* ID of the person.
- **location_id** (*int*) – The Birth *Location* ID of the person.
- **tag_ids** (*List[int]*) – A list of *Tag* IDs of the person.
- **blood_id** (*int*) – The *BloodType* ID of the person.
- **call_count** (*int*) – The number of times the person has been called.

:param : :type : returns: None

3.17 Group

class IreneAPIWrapper.models.**Group**(*group_id*, *name*, *date*, *description*, *company*, *display*, *website*, *social*, *media_count*, *tags*, *aliases*)

Represents a Group object.

A Group object inherits from *AbstractModel*.

Parameters

- **group_id** (*int*) – The group's unique ID.
- **name** (*str*) – The name of the group.

- **date** (*Date*) – The creation and disbandment of the group.
- **description** (*str*) – An overall description of the group.
- **company** (*Company*) – The company that owns the group.
- **display** (*Display*) – The display media (avatar & banner) for the group.
- **website** (*str*) – A custom website for the group.
- **social** (*Social*) – The social media associated with the group.
- **media_count** (*int*) – The media a group has.
- **tags** (List[*Tag*]) – The tags that affiliated with the group.
- **aliases** (List[*GroupAlias*]) – Aliases of the group.

id

The group's unique ID.

Type
int

name

The name of the group.

Type
str

date

The creation and disbandment of the group.

Type
Date

description

An overall description of the group.

Type
str

company

The company that owns the group.

Type
Company

display

The display media (avatar & banner) for the group.

Type
Display

website

A custom website for the group.

Type
str

social

The social media associated with the group.

Type
Social

`media_count`

The media a group has.

Type
int

`tags`

The tags that affiliated with the group.

Type
List[*Tag*]

`aliases`

Aliases of the group.

Type
List[*GroupAlias*]

`affiliations`

All affiliations that are associated with the Group.

Type
List[*Affiliation*]

`async static create`(**args*, ***kwargs*)

Create a Group object.

Returns
Group

`async delete`() → None

Delete the Group object from the database and remove it from cache.

Returns
None

`async static fetch`(*group_id: int*)

Fetch an updated Group object from the API.

Parameters
group_id – int The group's ID to fetch.

Returns
Group

`async static fetch_all`()

Fetch all groups.

`async static get`(*group_id: int*, *fetch=True*)

Get a *Group* object.

If the *Group* object does not exist in cache, it will fetch the name from the API. :param group_id: int

The ID of the *Group* to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Group

async static get_all()

Get all Group objects in cache.

Returns
dict_values[*Group*] All Group objects from cache.

async get_card(*markdown=False*, *extra=True*)

Get a list representing of the current object as a card.

Parameters
markdown – bool Whether the returned list should support markdown.

Returns
List[str] A list of strings for the card.

async static insert(*group_name: str*, *date_id: Optional[int] = None*, *description: Optional[str] = None*, *company_id: Optional[int] = None*, *display_id: Optional[int] = None*, *website: Optional[str] = None*, *social_id: Optional[int] = None*, *tag_ids: Optional[List[int]] = None*) → None

Insert a new group into the database.

Parameters

- **group_name** – str The group's name.
- **date_id** – int *Date* ID including the creation and disbandment dates.
- **description** – str Description of the overall group.
- **company_id** – int ID of the *Company* the group belongs to.
- **display_id** – int ID of the *Display* the group is associated with.
- **website** – str A custom website for the group.
- **social_id** – int ID of the *Social* the group has.
- **tag_ids** – List[int] A list of *Tag* IDs.

Returns
None

3.18 Fandom

class IreneAPIWrapper.models.**Fandom**(*group_id: int*, *fandom_name: str*, **args*, ***kwargs*)

Represents the fandom name of a *Group*.

A Fandom object inherits from *AbstractModel*.

Parameters

- **group_id** (*int*) – The *Group* ID.
- **fandom_name** (*str*) – The name of the fandom.

id

The *Group* ID associated with the fandom name.

Type
int

name

The name of the fandom.

Type
str

async static create(**args*, ***kwargs*)

Create a Fandom object.

Returns
Fandom

async delete() → None

Delete the Fandom object from the database and remove it from cache.

Returns
None

async static fetch(*group_id: int*)

Fetch an updated Fandom object from the API.

Parameters
group_id – int The group's ID to fetch a fandom for.

Returns
Fandom

async static fetch_all()

Fetch all fandoms.

async static get(*group_id: int*, *fetch=True*)

Get a Fandom object.

If the Fandom object does not exist in cache, it will fetch the name from the API. :param group_id: int

The ID of the group to get/fetch a fandom name for.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Fandom

async static get_all()

Get all Fandom objects in cache.

Returns
dict_values[*Fandom*] All Fandom objects from cache.

async static insert(*group_id*, *fandom_name*) → None

Insert a new fandom name into the database.

Parameters

- **group_id** – int The *Group* ID associated with the fandom name

- **fandom_name** – str The fandom name of the *Group*

Returns
None

3.19 Display

class IreneAPIWrapper.models.**Display**(*display_id*, *avatar:* MediaSource, *banner:* MediaSource, **args*, ***kwargs*)

Represents the images involved with an entity's profile such as an avatar or banner.

A Display object inherits from *AbstractModel*.

Parameters

- **display_id** (*int*) – The Affiliation id.
- **avatar** (*MediaSource*) – The person that is affiliated with a Group.
- **banner** (*MediaSource*) – The group that the Person is affiliated with.

id

The Display id.

Type
int

avatar

The person that is affiliated with a Group.

Type
MediaSource

banner

The group that the Person is affiliated with.

Type
MediaSource

async static create(**args*, ***kwargs*)

Create a Display object.

Returns
Display

async delete() → None

Delete the Display object from the database and remove it from cache.

async static fetch(*display_id: int*)

Fetch an updated Display object from the API.

Parameters
display_id – int The display's ID to fetch.

Returns
Optional[*Display*] The display object requested.

async static fetch_all()

Fetch all displays.

async static get(*display_id: int*, *fetch=True*)

Get a Display object.

If the Display object does not exist in cache, it will fetch the name from the API. :param display_id: int

The ID of the display to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Optional[*Display*] The display object requested.

async static get_all()

Get all Display objects in cache.

Returns
dict_values[*Display*] All Display objects from cache.

async get_card(*markdown=False*)

Get a list representing of the current object as a card.

Parameters
markdown – bool Whether the returned list should support markdown.

Returns
List[str] A list of strings for the card.

async static insert(*avatar: str*, *banner: str*) → None

Insert a new display into the database.

Parameters

- **avatar** – str The avatar for the entity.
- **banner** – str The banner for the entity.

Returns
None

3.20 Social

class IreneAPIWrapper.models.**Social**(*social_id*, *twitter*, *youtube*, *melon*, *instagram*, *vlive*, *spotify*, *fancafe*, *facebook*, *tiktok*)

Represents the social media sources for an entity.

A Social object inherits from *AbstractModel*.

Parameters

- **social_id** (*int*) – The Social ID
- **twitter** (*str*) – The Twitter code
- **youtube** (*str*) – The youtube code
- **melon** (*str*) – The melon code
- **instagram** (*str*) – The instagram code
- **vlive** (*str*) – The vlive code

- **spotify** (*str*) – The spotify code
- **fancafe** (*str*) – The fancafe code
- **facebook** (*str*) – The facebook code
- **tiktok** (*str*) – The tiktok code

id

The Social ID

Type
int

twitter

The Twitter code

Type
str

youtube

The youtube code

Type
str

melon

The melon code

Type
str

instagram

The instagram code

Type
str

vlive

The vlive code

Type
str

spotify

The spotify code

Type
str

fancafe

The fancafe code

Type
str

facebook

The facebook code

Type
str

tiktok

The tiktok code

Type
str

async static create(**args*, ***kwargs*)

Create an object.

async delete() → None

Delete the Social object from the database and remove it from cache.

Returns
None

async static fetch(*social_id: int*)

Fetch an updated Social object from the API.

Parameters
social_id – int The social's ID to fetch.

Returns
Social

async static fetch_all()

Fetch all socials.

async static get(*social_id: int*, *fetch=True*)

Get a Social object.

If the Social object does not exist in cache, it will fetch the name from the API. :param social_id: int

The ID of the name to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Social

async static get_all()

Get all Social objects in cache.

Returns
dict_values[*Social*] All Social objects from cache.

async get_card(*markdown=False*)

Get a list representing of the current object as a card.

Parameters
markdown – bool Whether the returned list should support markdown.

Returns
List[str] A list of strings for the card.

async static insert(*twitter*, *youtube*, *melon*, *instagram*, *vlive*, *spotify*, *fancafe*, *facebook*, *tiktok*) → None

Insert a new social into the database.

Parameters

- **twitter** (*str*) – Twitter Code

- **youtube** (*str*) – Youtube Code
- **melon** (*str*) – Melon code
- **instagram** (*str*) – Instagram code
- **vlive** (*str*) – Vlive code
- **spotify** (*str*) – Spotify code
- **fancafe** (*str*) – fancafe code
- **facebook** (*str*) – Facebook code
- **tiktok** (*str*) – Tiktok code

:param : :type : returns: None

3.21 Position

class IreneAPIWrapper.models.**Position**(*position_id*, *name*)

Represents a position or status.

A Position object inherits from *AbstractModel*.

Parameters

- **position_id** (*int*) – The Position id.
- **name** (*str*) – The position's name.

id

The Position id.

Type
int

name

The position's name.

Type
str

async static create(**args*, ***kwargs*)

Create a Position object.

Returns
Position

async delete() → None

Delete the Position object from the database and remove it from cache.

Returns
None

async static fetch(*position_id: int*)

Fetch an updated Position object from the API.

Parameters
position_id – int The position's ID to fetch.

Returns
Position

`async static fetch_all()`

Fetch all positions.

`async static get(position_id: int, fetch=True)`

Get a Position object.

If the Position object does not exist in cache, it will fetch the name from the API. :param position_id: int

The ID of the name to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Position

`async static get_all()`

Get all Position objects in cache.

Returns
dict_values[*Position*] All Position objects from cache.

`async static insert(name)` → None

Insert a new position into the database.

Parameters
name – str The position name.

Returns
None

3.22 Company

`class IreneAPIWrapper.models.Company(company_id, name, description, date, *args, **kwargs)`

Represents the business/company that exists for several entities.

A Company object inherits from *AbstractModel*.

Parameters

- **company_id** (*int*) – The company's unique ID
- **name** (*str*) – The company's name.
- **description** (*str*) – A general description of the company as a whole.
- **date** (*Date*) – The Date object that involves the creation and retirement of the company.

`id`

The company's unique ID

Type
int

name

The company's name.

Type
str

description

A general description of the company as a whole.

Type
str

date

The Date object that involves the creation and retirement of the company.

Type
Date

async static create(**args*, ***kwargs*)

Create a Company object.

Returns
Company

async delete()

Delete the Company object from the database and remove it from cache.

async static fetch(*company_id: int*)

Fetch an updated Company object from the API.

Parameters
company_id – int The company's ID to fetch.

Returns
Optional[*Company*] The company object requested.

async static fetch_all()

Fetch all companies.

async static get(*company_id: int*, *fetch=True*)

Get a Company object.

If the Company object does not exist in cache, it will fetch the name from the API. :param company_id: int

The ID of the company to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Optional[*Company*] The company object requested.

async static get_all()

Get all Company objects in cache.

Returns
dict_values[*Company*] All Company objects from cache.

async get_card(*markdown=False*)

Get a list representing of the current object as a card.

Parameters
markdown – bool Whether the returned list should support markdown.

Returns
List[str] A list of strings for the card.

async static insert(*company_name*, *description*, *date:* Date) → None

Insert a new company into the database.

Parameters

- **company_name** – str The company's name.
- **description** – A description of the company as a whole.
- **date** – The *Date* object for the creation and retirement of the company.

Returns
None

3.23 PersonAlias

class IreneAPIWrapper.models.**PersonAlias**(*alias_id*, *alias_name*, *person_id*, *guild_id*)

Represents the alias of a *Person*.

A PersonAlias object inherits from *Alias* which inherits from *AbstractModel*.

Parameters

- **alias_id** (*int*) – The Alias id.
- **alias_name** (*str*) – The alias name.
- **person_id** (*int*) – The ID of the *Person* the alias is referring to.
- **guild_id** (*Optional[int]*) – A guild ID that owns the alias if there is one.

id

The Alias id.

Type
int

name

The alias name.

Type
str

_obj_id

The *Person* ID the alias is referring to. Used for Abstraction.

Type
int

person_id

The *Person* ID the alias is referring to.

Type
int

guild_id

A guild ID that owns the alias if there is one.

Type
Optional[int]

async static create(**args*, ***kwargs*)

Create a PersonAlias object.

Returns
PersonAlias

async delete() → None

Delete the PersonAlias object from the database and remove it from cache.

Returns
None

async static fetch(*person_alias_id: int*)

Fetch an updated PersonAlias object from the API.

Parameters
person_alias_id – int The person alias's ID to fetch.

Returns
PersonAlias

async static fetch_all()

Fetch all person aliases.

async static get(*person_alias_id: int*, *fetch=True*)

Get a PersonAlias object.

If the PersonAlias object does not exist in cache, it will fetch the name from the API. :param person_alias_id: int

The ID of the name to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
PersonAlias

async static get_all()

Get all PersonAlias objects in cache.

Returns
dict_values[*PersonAlias*] All PersonAlias objects from cache.

async static insert(*person_id: int*, *alias: str*, *guild_id: Optional[int] = None*) → bool

Insert a new PersonAlias into the database.

Parameters

- **person_id** – int The *Group*'s ID.
- **alias** – str The alias of the *Person* to add.
- **guild_id** – Optional[int] A guild that owns this alias.

Returns
bool Whether the PersonAlias was added to the existing objects as well as inserted into the DB.

3.24 GroupAlias

class IreneAPIWrapper.models.**GroupAlias**(*alias_id*, *alias_name*, *group_id*, *guild_id*)

Represents the alias of a *Group*.

A GroupAlias object inherits from *Alias* which inherits from *AbstractModel*.

Parameters

- **alias_id** (*int*) – The Alias id.
- **alias_name** (*str*) – The alias name.
- **group_id** (*int*) – The ID of the *Group* the alias is referring to.
- **guild_id** (*Optional[int]*) – A guild ID that owns the alias if there is one.

id

The Alias id.

Type
int

name

The alias name.

Type
str

_obj_id

The *Group* ID the alias is referring to. Used for Abstraction.

Type
int

group_id

The *Group* ID the alias is referring to.

Type
int

guild_id

A guild ID that owns the alias if there is one.

Type
Optional[int]

async static create(**args*, ***kwargs*)

Create a GroupAlias object.

Returns
GroupAlias

async delete() → None

Delete the GroupAlias object from the database and remove it from cache.

Returns
None

async static fetch(*group_alias_id: int*)

Fetch an updated GroupAlias object from the API.

Parameters
group_alias_id – int The group alias's ID to fetch.

Returns
GroupAlias

async static fetch_all()

Fetch all group aliases.

async static get(*group_alias_id: int*, *fetch=True*)

Get a GroupAlias object.

If the GroupAlias object does not exist in cache, it will fetch the name from the API. :param group_alias_id: int

The ID of the group alias to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
GroupAlias

async static get_all()

Get all GroupAlias objects in cache.

Returns
dict_values[*GroupAlias*] All GroupAlias objects from cache.

async static insert(*group_id: int*, *alias: str*, *guild_id: Optional[int] = None*) → bool

Insert a new GroupAlias into the database.

Parameters

- **group_id** – int The *Group*'s ID.
- **alias** – str The alias of the *Group* to add.
- **guild_id** – Optional[int] A guild that owns this alias.

Returns
bool Whether the GroupAlias was added to the existing objects as well as inserted into the DB.

3.25 CallBack

class IreneAPIWrapper.models.**CallBack**(*callback_type: str = 'request'*, *request: Optional[dict] = None*)

Represents a CallBack from the API (Request & Response).

Parameters

- **callback_type** (*str*) – The type of callback. Can be 'request' or 'disconnect'.
- **request** (*Optional[dict]*) – A request if it's already known.

id

Is the CallBack ID. Consists of a random integer from 0 to 50,000 concatenated with the current timestamp.

Type
int

type

The callback type. Can be 'request' or 'disconnect'.

Type
str

_creation_time

The creation time of the object.

Type
datetime

done

Whether a response has been received.

Type
bool

request

The request to be sent to the API.

Type
Optional[dict]

response

The response received from the API.

Type
Optional[dict]

_completion_time

The time the response from the API was received.

Type
datetime

_expected_result

Used for testing expected responses from the API.

Type
Optional[dict]

set_as_done() → None

Set the current callback as finished.

Returns
None

async wait_for_completion(*timeout: Optional[int] = None*) → bool

Waits for a response from the API.

Parameters
timeout – Optional[int] Seconds before no longer waiting for the response. (No timeout by default.)

Returns
True Only returns when there is a response from the API.

CHAPTER

FOUR

DATA MODELS

4.1 Timeline

class IreneAPIWrapper.models.**Timeline**(*results: List[dict]*, **args*, ***kwargs*)

Represents a Twitter Account's Timeline and contains a list of *Tweet* objects.

Parameters

results (*List[dict]*) – The results from the API containing a list of dicts that contain the ID and text of the Tweet.

tweets

A list of tweets.

Type

List[*Tweet*]

latest_tweet

The latest tweet.

Type

Optional[*Tweet*]

property latest_tweet: Optional[*Tweet*]

Get the latest tweet.

update_tweets(*results: List[dict]*)

Update the list of tweets with information from the API.

4.2 Subscription

class IreneAPIWrapper.models.**Subscription**(*account_id: Union[int, str]*, *account_name: str*, *followed: Optional[List[*Channel*]] = None*, *mention_roles: Optional[Dict[*Channel*, int]] = None*)

Abstract Subscription Class for a service account being followed by a user, guild, or channel.

Parameters

- **account_id** (*int*) – The account's ID.
- **account_name** (*str*) – The account's name.
- **followed** (Optional[List[*Channel*]]) – The *Channel* objects following the service account.

id

Account ID.

Type
int

name

The account's name.

Type
str

_followed

The list of *Channel* objects followed to the service account.

Type
List[*Channel*]

_mention_roles

Channel objects associated with role ids to mention on updates.

Type
Dict[*Channel*, int]

check_subscribed(*channels: List[*Channel*]*) → List[*Channel*]

Checks which :ref:`Channel`s are subscribed to the current subscription account from a selection of channels.

Parameters
channels – List[*Channel*]

:returns List[*Channel*]
A list of :ref:`Channel`s from the channels provided that are subscribed.

async get_role_id(*channel:* Channel)

Get the role id to mention of a channel.

async subscribe(*channel:* Channel, *role_id: Optional[int] = None*) → None

Subscribe to a channel. :param role_id: The role id to notify. :param channel: *Channel* :return: None

async unsubscribe(*channel:* Channel) → None

Unsubscribe from an account. :param channel: *Channel* :return: None

4.3 TwitterAccount

class IreneAPIWrapper.models.**TwitterAccount**(*account_id: int*, *account_name: str*, *channels_following: Optional[List[*Channel*]] = None*, *mention_roles: Optional[Dict[*Channel*, int]] = None*)

Represents a Twitter Account.

A TwitterAccount object inherits from *Subscription*.

Parameters

- **account_id** (*int*) – The Twitter Account's ID.
- **account_name** (*str*) – The account's username.

- **channels_following** (Optional[List[*Channel*]]) – List of *Channel* objects that are following the Twitter Account.
- **mention_roles** (Optional[Dict[*Channel*, int]]) – Roles that are to be mentioned in a discord channel.

id

Account ID.

Type
int

name

The account's name.

Type
str

latest_tweet

The latest tweet on the Twitter Account.

Type
Optional[*Tweet*]

async static check_user_exists(*username*) → bool

Check if a Twitter username exists.

Parameters
username – The Twitter display username.

Returns
bool Whether the username exists.

async static create(**args*, ***kwargs*)

Create a TwitterAccount object.

If several rows containing the same accounts are being passed in, use create_bulk instead for proper optimization. This will happen by default in a fallback if multiple rows are detected.

Returns
Union[*TwitterAccount*, Optional[List[*TwitterAccount*]]]

async static create_bulk(*list_of_dicts: List[dict]*)

Bulk create TwitterAccount objects.

Parameters
list_of_dicts – List[dict] A list of dictionaries.

Returns
Optional[List[*TwitterAccount*]]

async delete()

Delete Twitter account and it's followings.

async static fetch(*username: str*)

Fetch an updated TwitterAccount object from the API.

Parameters
username – int The Twitter account username to fetch.

Returns
Optional[*TwitterAccount*] The TwitterAccount object requested.

async static fetch_all()

Fetch all TwitterAccounts objects from the database.

Returns
List[*TwitterAccount*] A list of TwitterAccount objects.

async fetch_timeline() → *Timeline*

Fetch the latest tweets for this account from Twitter

async static get(*username: Optional[str] = None*, *fetch=True*) → Optional[*Subscription*]

Get a TwitterAccount instance from cache or fetch it from the api.

Parameters
- **username** – str The username of the Twitter account.
- **fetch** – bool Whether to fetch from the API if not found in cache.

Returns
Optional[*TwitterAccount*] The TwitterAccount object.

async static get_all()

Get all TwitterAccount objects in cache.

Returns
dict_values[*TwitterAccount*] All TwitterAccount objects from cache.

async static get_twitter_id(*username*) → Optional[int]

Get the Twitter account id of a username if it exists.

Parameters
username – The Twitter display username.

Returns
Optional[int] The account ID.

async static insert(*username: str*, *guild_id: int*, *channel_id: int*, *role_id: Optional[int]*)

Insert a new TwitterAccount into the database.

Parameters
- **username** – int The TwitterAccount username.
- **guild_id** – int The first guild ID that is subscribing.
- **channel_id** – int The first channel id that is subscribing.
- **role_id** – Optional[int] A role to notify.

Returns
TwitterAccount The TwitterAccount object.

property latest_tweet: Optional[*Tweet*]

Get the latest tweet.

async static subbed_in(*guild_id*)

Get the Twitter channels subscribed to in a Guild.

Parameters
guild_id – The guild ID.

Returns
Optional[List[*TwitterAccount*]]

async subscribe(*channel:* Channel, *role_id=None*)

Have a channel subscribe to the account if it is not already.

Parameters

- **channel** – The channel to subscribe to the account.
- **role_id** – The role id to mention.

async unsubscribe(*channel:* Channel)

Have a channel unsubscribe from the account if it is not already.

Parameters
channel – *Channel* The channel to unsubscribe from the account.

4.4 TwitchAccount

class IreneAPIWrapper.models.**TwitchAccount**(*username: str*, *channels_following: Optional[List[*Channel*]] = None*, *mention_roles: Optional[Dict[*Channel*, int]] = None*)

Represents a Twitch Account.

A TwitchAccount object inherits from *Subscription*.

Parameters

- **username** (*str*) – The twitch account username
- **channels_following** (Optional[List[*Channel*]]) – The channels following the Twitch account
- **mention_roles** (*Optional[Dict[*Channel*, int]]*) – The role ids of channels that need mentioning on updates.

async check_live() → bool

Check if the current twitch account is live.

Returns
bool

async static check_live_bulk(*accounts: List[*AbstractModel*]*) → Dict[str, bool]

A list of Twitch accounts.

Parameters
accounts – List[*TwitchAccount*] A list of twitch accounts.

Returns
Dict[str, bool] A dictionary with the key as the username and the value if they are live.

async static check_user_exists(*username*) → bool

Check if a twitch username exists.

Parameters
username – The twitch display or login username.

Returns
bool Whether the username exists.

async static create(**args*, ***kwargs*)

Create a TwitchAccount object.

If several rows containing the same accounts are being passed in, use create_bulk instead for proper optimization. This will happen by default in a fallback if multiple rows are detected.

Returns
TwitchAccount

async static create_bulk(*list_of_dicts: List[dict]*)

Bulk create TwitchAccount objects.

Parameters
list_of_dicts – List[dict] A list of dictionaries.

Returns
Optional[List[*TwitchAccount*]]

async static fetch(*username: str*)

Fetch an updated TwitchAccount object from the API.

Parameters
username – int The Twitch account username to fetch.

Returns
Optional[*TwitchAccount*] The TwitchAccount object requested.

async static fetch_all()

Fetch all TwitchAccount objects.

Note: TwitchAccount objects are added to cache on creation.

---

async static get(*username: str*, *fetch=True*)

Get a TwitchAccount object.

If the TwitchAccount object does not exist in cache, it will fetch the id from the API. :param username: str

The twitch account username.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Optional[*TwitchAccount*] The TwitchAccount object requested.

async static get_all()

Get all TwitchAccount objects in cache.

Returns
dict_values[*TwitchAccount*] All TwitchAccount objects from cache.

async get_posted() → List[*Channel*]

Get a list of channels that have already posted to discord. :return: List[*Channel*]

async static insert(*username: str*, *channel_id: int*, *role_id: Optional[int]*)

Insert a new TwitchAccount into the database.

Parameters

- **username** – int The Twitch Account username.

- **channel_id** – int The first channel id that is subscribing.
- **role_id** – Optional[int] A role to notify.

Returns
TwitchAccount The TwitchAccount object.

async static subbed_in(*guild_id*)

Get the twitch channels subscribed to in a Guild.

Parameters
guild_id – The guild ID.

Returns
Optional[List[*TwitchAccount*]]

async subscribe(*channel:* Channel, *role_id: Optional[int] = None*)

Subscribe to a channel. :param role_id: The role id to notify. :param channel: *Channel* :return: None

async unsubscribe(*channel:* Channel)

Have a channel unsubscribe from the account if it is not already.

Parameters
channel – *Channel* The channel to unsubscribe from the account.

async update_posted(*channel_ids: List[int]*, *posted: bool*) → None

Update the media and status ids for the game in the database.

Parameters

- **channel_ids** – List[int] All channel IDs that need the posted attribute changed.
- **posted** – bool Whether the update has been posted to the channels.

Returns
None

4.5 Tweet

class IreneAPIWrapper.models.**Tweet**(**args*, ***kwargs*)

Represents a Twitter Account's Tweet.

id

The Tweet ID.

Type
int

content

The Tweet's contents

Type
str

property is_reply: bool

Whether the tweet is a reply (sometimes)

..Note:: Twitter sucks and doesn't let us know if it's a reply, so here we are filtering by if the user @ed someone at the start of their tweet…

property is_retweet: bool

Whether the tweet is a retweet.

4.6 Access

class IreneAPIWrapper.models.Access(*access_id: int*)

Represents the Access level of the API.

Please note that these Access values are consistent across the API as well.

id

The representative Access ID.

Type
int

4.7 BloodType

class IreneAPIWrapper.models.BloodType(*blood_id*, *name*)

Represents a BloodType.

A BloodType object inherits from *AbstractModel*.

Please note that the blood types are metadata. While it may be possible to delete/remove blood types from the API, it will not be possible in this wrapper to avoid unnecessary changes.

Note: Possible Blood Types: O- O+ A- A+ B- B+ AB- AB+

id

The blood type's id.

Type
int

name

The name of the blood type.

Type
str

async static create(**args*, ***kwargs*)

Create an object.

async static fetch(*blood_id: int*)

Fetch an updated BloodType object from the API.

Note: BloodType objects are added to cache on creation.

Parameters
blood_id – int The blood's ID to fetch.

Returns
Optional[*BloodType*] The blood type object requested.

`async static fetch_all()`
Fetch all blood types.

Note: BloodType objects are added to cache on creation.

`async static get(blood_id: int, fetch=True)`
Get a BloodType object.

If the BloodType object does not exist in cache, it will fetch the name from the API.

Parameters

- **blood_id** – int The ID of the blood type to get/fetch.
- **fetch** – bool Whether to fetch from the API if not found in cache.

Returns
Optional[*BloodType*] The blood type object requested.

`async static get_all()`
Get all BloodType objects in cache.

Returns
dict_values[*BloodType*] All BloodType objects from cache.

4.8 Name

`class IreneAPIWrapper.models.Name(name_id, first, last)`
Represents names for an entity that may have several types.

A Name object inherits from *AbstractModel*.

Parameters

- **name_id** (*int*) – The Name id.
- **first** (*str*) – First part of the name.
- **last** (*str*) – Last part of the name.

`id`
The Name id.

Type
int

`first`
First part of the name.

Type
str

`last`

Last part of the name.

Type
str

`async static create`(**args*, ***kwargs*)

Create a name object.

Returns
Name

`async delete`() → None

Delete the Name object from the database and remove it from cache.

Returns
None

`async static fetch`(*name_id: int*)

Fetch an updated Name object from the API.

Parameters
name_id – int The name's ID to fetch.

Returns
Name

`async static fetch_all`()

Fetch all names.

`async static get`(*name_id: int*, *fetch=True*)

Get a Name object.

If the Name object does not exist in cache, it will fetch the name from the API. :param name_id: int

The ID of the name to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

Returns
Name

`async static get_all`()

Get all Name objects in cache.

Returns
dict_values[*Name*] All Name objects from cache.

`async get_card`(*markdown=False*, *extra=True*)

Get a list representing of the current object as a card.

Parameters
markdown – bool Whether the returned list should support markdown.

Returns
List[str] A list of strings for the card.

async static insert(*first*, *last*) → None

Insert a new name into the database.

Parameters

- **first** – str The first part of the name.
- **last** – str The last part of the name.

Returns
None

4.9 Tag

class IreneAPIWrapper.models.**Tag**(*tag_id*, *name*, **args*, ***kwargs*)

Represents a tag that describes an entity.

A Tag object inherits from *AbstractModel*.

Parameters

- **tag_id** (*int*) – The Tag's id.
- **name** (*str*) – The tag's name.

id

The Tag id.

Type
int

name

The tag name.

Type
str

async static create(**args*, ***kwargs*)

Create a Tag object.

Returns
Tag

async delete() → None

Delete the Tag object from the database and remove it from cache.

Returns
None

async static fetch(*tag_id: int*)

Fetch an updated Tag object from the API.

Parameters
tag_id – int The tag's ID to fetch.

async static fetch_all()

Fetch all tags.

async static get(*tag_id: int*, *fetch=True*)

Get a Tag object.

If the Tag object does not exist in cache, it will fetch the tag from the API. :param tag_id: int

The ID of the tag to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

async static get_all()

Get all Tag objects in cache.

Returns
dict_values[*Tag*] All Tag objects from cache.

async static insert(*tag_name*) → None

Insert a new Tag into the database.

Parameters
tag_name –

Returns
None

4.10 Date

class IreneAPIWrapper.models.**Date**(*date_id: int*, *start_date: str*, *end_date: str*, **args*, ***kwargs*)

Represents a starting and end date of an entity.

A Date object inherits from *AbstractModel*.

Parameters

- **date_id** (*int*) – The Date ID.
- **start_date** (*str*) – The start date.
- **end_date** (*str*) – The end date.

id

The Date id.

Type
int

start

The start date.

Type
str

end

The end date.

Type
str

async static create(**args*, ***kwargs*)

Create a Date object.

Returns
Date

async delete() → None

Delete the Date object from the database and remove it from cache.

Returns
None

async static fetch(*date_id: int*)

Fetch an updated Date object from the API.

Parameters
date_id – int The date's ID to fetch.

Returns
Optional[*Date*] The date object requested.

async static fetch_all()

Fetch all dates.

async static get(*date_id: int*, *fetch=True*)

Get a Date object.

If the Date object does not exist in cache, it will fetch the date from the API.

Parameters

- **date_id** – int The ID of the date to get/fetch.
- **fetch** – bool Whether to fetch from the API if not found in cache.

Returns
Optional[*Date*] The date object requested.

async static get_all()

Get all Date objects in cache.

Returns
dict_values[*Date*] All Date objects from cache.

async get_card(*markdown=False*)

Get a list representing of the current object as a card.

Parameters
markdown – bool Whether the returned list should support markdown.

Returns
List[str] A list of strings for the card.

async static insert(*start_date*, *end_date=None*) → int

Insert a new date into the database.

Parameters

- **start_date** – Union[str, Datetime] Datetime or string object in '%Y-%m-%d %H:%M:%S.%f' format (equivalent to datetime.now()). Is the start date.
- **end_date** – Union[str, Datetime] Datetime or string object in '%Y-%m-%d %H:%M:%S.%f' format (equivalent to datetime.now()). Is the end date.

Returns
int The Date id

`async update_end_date`(*end_date*) → None

Update the end date.

Parameters
end_date – Union[str, Datetime] Datetime or string object in '%Y-%m-%d %H:%M:%S.%f' format (equivalent to datetime.now()). Is the end date.

Returns
None

4.11 Mode

`class IreneAPIWrapper.models.Mode`(*access_id: int*, *name: str*)

Represents the Modes for certain entities.

Please note that these Mode values are consistent across the API as well.

id

The representative mode ID.

Type
int

name

The name of the mode.

Type
str

4.12 Preload

`class IreneAPIWrapper.models.Preload`(*force: bool = True*)

Represents which attributes are loaded up on client startup.

force

Whether to make sure all cache is preloaded. (Defaults to True)

Type
bool

tags

Whether to preload all cache for tags (Defaults to True).

Type
bool

person_aliases

Whether to preload all cache for person_aliases (Defaults to True).

Type
bool

`group_aliases`
: Whether to preload all cache for group_aliases (Defaults to True).

  Type
  bool

`persons`
: Whether to preload all cache for persons (Defaults to True).

  Type
  bool

`groups`
: Whether to preload all cache for groups (Defaults to True).

  Type
  bool

`twitter_accounts`
: Whether to preload all cache for twitter_accounts (Defaults to True).

  Type
  bool

`users`
: Whether to preload all cache for users (Defaults to False).

  Type
  bool

`guilds`
: Whether to preload all cache for guilds (Defaults to False).

  Type
  bool

`affiliations`
: Whether to preload all cache for affiliations (Defaults to True).

  Type
  bool

`bloodtypes`
: Whether to preload all cache for bloodtypes (Defaults to True).

  Type
  bool

`media`
: Whether to preload all cache for media (Defaults to True).

  Type
  bool

`displays`
: Whether to preload all cache for displays (Defaults to True).

  Type
  bool

`companies`
Whether to preload all cache for companies (Defaults to True).

Type
bool

`dates`
Whether to preload all cache for dates (Defaults to True).

Type
bool

`locations`
Whether to preload all cache for locations (Defaults to True).

Type
bool

`positions`
Whether to preload all cache for positions (Defaults to True).

Type
bool

`socials`
Whether to preload all cache for socials (Defaults to True).

Type
bool

`names`
Whether to preload all cache for names (Defaults to True).

Type
bool

`fandoms`
Whether to preload all cache for fandoms (Defaults to True).

Type
bool

`channels`
Whether to preload all text channels (Defaults to False).

Type
bool

`twitch_subscriptions`
Whether to preload all twitch subscriptions (Defaults to False).

Type
bool

`twitter_subscriptions`
Whether to preload all twitter subscriptions (Defaults to False).

Type
bool

`languages`
: Whether to preload all languages (Defaults to True)

Type
bool

`eight_ball_responses`
: Whether to preload all 8ball responses (Defaults to True)

Type
bool

`notifications`
: Whether to preload all user notifications (Defaults to True)

Type
bool

`interactions`
: Whether to preload all interactions (Defaults to True)

Type
bool

`auto_media`
: Whether to preload all auto media (Defaults to True)

Type
bool

`reaction_role_messages`
: Whether to preload all reaction role messages (Defaults to True)

Type
bool

4.13 Difficulty

class IreneAPIWrapper.models.**Difficulty**(*diff_id: int*, *name: str*)

Represents the Difficulty levels.

Please note that these Difficulty values are consistent across the API as well.

`id`
: The difficulty ID.

Type
int

`name`
: The difficulty name.

Type
str

4.14 Location

class IreneAPIWrapper.models.**Location**(*location_id*, *country*, *city*)

Represents a location.

A Location object inherits from *AbstractModel*.

Parameters

- **location_id** (*int*) – The Affiliation id.
- **country** (*str*) – The country's name.
- **city** (*str*) – The city's name.

id

The Location id.

Type
int

country

The country's name.

Type
str

city

The city's name.

Type
str

async static create(**args*, ***kwargs*)

Create a Location object.

Returns
Location

async delete() → None

Delete the Location object from the database and remove it from cache.

Returns
None

async static fetch(*location_id: int*)

Fetch an updated Location object from the API.

Parameters
location_id – int The location's ID to fetch.

async static fetch_all()

Fetch all locations.

async static get(*location_id: int*, *fetch=True*)

Get a Location object.

If the Location object does not exist in cache, it will fetch the name from the API. :param location_id: int

The ID of the location to get/fetch.

Parameters
fetch – bool Whether to fetch from the API if not found in cache.

async static get_all()
Get all Location objects in cache.

Returns
dict_values[*Location*] All Location objects from cache.

async static insert(*country*, *city*) → None
Insert a new location into the database.

Parameters

- **country** – The country's name.
- **city** – The city's name.

Returns
None

4.15 UserStatus

class IreneAPIWrapper.models.**UserStatus**(*status_id: int*, *user_id: int*, *score: int*, **args*, ***kwargs*)
Represents a user's status in a game.

A UserStatus object inherits from *AbstractModel*.

Parameters

- **status_id** (*int*) – The status ID.
- **user_id** (*int*) – The user id.
- **score** (*int*) – The score.

id
The status ID.

Type
int

user_id
The user id.

Type
int

score
The score.

Type
int

async static create(**args*, ***kwargs*)
Create a *UserStatus* object.

Returns
UserStatus

async delete() → None

Delete the Status object from the database and remove it from cache.

Returns
None

async static fetch(*status_id: int*)

Fetch an updated UserStatus object from the API.

Parameters
status_id – int The status ID to fetch.

Returns
Optional[*UserStatus*] The user status object requested.

async static fetch_all()

Fetch all statuses.

async static get(*status_id: int*, *fetch=True*)

Get a Status object.

If the Status object does not exist in cache, it will fetch the date from the API.

Parameters

- **status_id** – int The ID of the status to get/fetch.
- **fetch** – bool Whether to fetch from the API if not found in cache.

Returns
Optional[*UserStatus*] The UserStatus object requested.

async static get_all()

Get all UserStatus objects in cache.

Returns
dict_values[*UserStatus*] All UserStatus objects from cache.

async static insert(*user_id*, *score=0*) → int

Insert a new status into the database.

Parameters

- **user_id** – int The user's ID.
- **score** – int The score of the player.

Returns
int The Status id

async update_score(*score: Optional[int] = None*) → None

Update the score.

Parameters
score – int The player score.

Returns
None

CHAPTER

FIVE

EXCEPTIONS

5.1 InvalidToken

exception IreneAPIWrapper.exceptions.**InvalidToken**

An Exception Raised When an Invalid Token was Supplied.

5.2 APIError

exception IreneAPIWrapper.exceptions.**APIError**(*callback:* CallBack, *error_msg=None*, *detailed_report=False*)

An Exception Raised When the API returned an error.

5.3 Empty

exception IreneAPIWrapper.exceptions.**Empty**

An exception caused when an iterable is empty.

5.4 IncorrectNumberOfItems

exception IreneAPIWrapper.exceptions.**IncorrectNumberOfItems**(*msg*)

An Exception caused when there is not enough or too much of something (for example arguments).

5.5 FailedObjectCreation

exception IreneAPIWrapper.exceptions.**FailedObjectCreation**(*callback*)

An exception caused when objects failed to properly create.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

Y