

---

# IreneAPIWrapper

MuJyKun

Feb 02, 2023



## CONTENTS:

<b>1</b>	<b>Client</b>	<b>1</b>
<b>2</b>	<b>Abstract Base Classes</b>	<b>3</b>
2.1	AbstractModel . . . . .	3
2.2	MediaSource . . . . .	4
2.3	Alias . . . . .	4
2.4	File . . . . .	5
2.5	Receiver . . . . .	5
<b>3</b>	<b>API Models</b>	<b>7</b>
3.1	Channel . . . . .	7
3.2	Interaction . . . . .	8
3.3	InteractionType . . . . .	9
3.4	Notification . . . . .	10
3.5	BiasGame . . . . .	12
3.6	GuessingGame . . . . .	13
3.7	UnscrambleGame . . . . .	15
3.8	Wolfram . . . . .	17
3.9	EightBallResponse . . . . .	17
3.10	PackMessage . . . . .	19
3.11	Language . . . . .	20
3.12	Guild . . . . .	21
3.13	Media . . . . .	26
3.14	User . . . . .	29
3.15	Affiliation . . . . .	31
3.16	Person . . . . .	33
3.17	Group . . . . .	36
3.18	Fandom . . . . .	39
3.19	Display . . . . .	41
3.20	Social . . . . .	42
3.21	Position . . . . .	45
3.22	Company . . . . .	46
3.23	PersonAlias . . . . .	48
3.24	GroupAlias . . . . .	50
3.25	CallBack . . . . .	52
<b>4</b>	<b>Data Models</b>	<b>55</b>
4.1	Timeline . . . . .	55
4.2	Subscription . . . . .	55
4.3	TwitterAccount . . . . .	56

4.4	TwitchAccount . . . . .	59
4.5	Tweet . . . . .	61
4.6	Access . . . . .	62
4.7	BloodType . . . . .	62
4.8	Name . . . . .	63
4.9	Tag . . . . .	65
4.10	Date . . . . .	66
4.11	Mode . . . . .	68
4.12	Preload . . . . .	68
4.13	Difficulty . . . . .	71
4.14	Location . . . . .	72
4.15	UserStatus . . . . .	73
<b>5</b>	<b>Exceptions</b>	<b>75</b>
5.1	InvalidToken . . . . .	75
5.2	APIError . . . . .	75
5.3	Empty . . . . .	75
5.4	IncorrectNumberOfItems . . . . .	75
5.5	FailedObjectCreation . . . . .	75
<b>6</b>	<b>Indices and tables</b>	<b>77</b>
	<b>Python Module Index</b>	<b>79</b>
	<b>Index</b>	<b>81</b>

## 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.

## 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.

## 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.



## 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) → NoneSubscribe to a channel. :param role\_id: The role id to notify. :param channel: *Channel* :return: None**async unsubscribe**(channel: *Channel*) → NoneUnsubscribe 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 data 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

## 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.





## INDICES AND TABLES

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



## PYTHON MODULE INDEX

### i

`IreneAPIWrapper.models.base.receiver`, [5](#)



## Symbols

`_completion_time` (*IreneAPIWrapper.models.CallBack attribute*), 52  
`_creation_time` (*IreneAPIWrapper.models.CallBack attribute*), 52  
`_expected_result` (*IreneAPIWrapper.models.CallBack attribute*), 52  
`_followed` (*IreneAPIWrapper.models.Subscription attribute*), 56  
`_mention_roles` (*IreneAPIWrapper.models.Subscription attribute*), 56  
`_obj_id` (*IreneAPIWrapper.models.Alias attribute*), 5  
`_obj_id` (*IreneAPIWrapper.models.GroupAlias attribute*), 50  
`_obj_id` (*IreneAPIWrapper.models.PersonAlias attribute*), 48

## A

`AbstractModel` (*class in IreneAPIWrapper.models*), 3  
`Access` (*class in IreneAPIWrapper.models*), 62  
`add_and_wait()` (*IreneAPIWrapper.models.IreneAPIClient method*), 2  
`add_prefix()` (*IreneAPIWrapper.models.Guild method*), 24  
`add_to_queue()` (*IreneAPIWrapper.models.IreneAPIClient method*), 2  
`add_token()` (*IreneAPIWrapper.models.User method*), 29  
`Affiliation` (*class in IreneAPIWrapper.models*), 31  
`affiliation` (*IreneAPIWrapper.models.Media attribute*), 26  
`affiliations` (*IreneAPIWrapper.models.Group attribute*), 38  
`affiliations` (*IreneAPIWrapper.models.Person attribute*), 35  
`affiliations` (*IreneAPIWrapper.models.Preload attribute*), 69  
`afk_timeout` (*IreneAPIWrapper.models.Guild attribute*), 22  
`Alias` (*class in IreneAPIWrapper.models*), 4  
`aliases` (*IreneAPIWrapper.models.Group attribute*), 38  
`aliases` (*IreneAPIWrapper.models.Person attribute*), 35

`APIError`, 75

`auto_media` (*IreneAPIWrapper.models.Preload attribute*), 71

`avatar` (*IreneAPIWrapper.models.Display attribute*), 41

## B

`banner` (*IreneAPIWrapper.models.Display attribute*), 41  
`banner` (*IreneAPIWrapper.models.Guild attribute*), 22  
`BiasGame` (*class in IreneAPIWrapper.models*), 12  
`blood_type` (*IreneAPIWrapper.models.Person attribute*), 34  
`BloodType` (*class in IreneAPIWrapper.models*), 62  
`bloodtypes` (*IreneAPIWrapper.models.Preload attribute*), 69  
`boosts` (*IreneAPIWrapper.models.Guild attribute*), 23

## C

`call_count` (*IreneAPIWrapper.models.Person attribute*), 34  
`CallBack` (*class in IreneAPIWrapper.models*), 52  
`category_count` (*IreneAPIWrapper.models.Guild attribute*), 23  
`Channel` (*class in IreneAPIWrapper.models*), 7  
`channels` (*IreneAPIWrapper.models.Preload attribute*), 70  
`check_live()` (*IreneAPIWrapper.models.TwitchAccount method*), 59  
`check_live_bulk()` (*IreneAPIWrapper.models.TwitchAccount static method*), 59  
`check_subscribed()` (*IreneAPIWrapper.models.Subscription method*), 56  
`check_user_exists()` (*IreneAPIWrapper.models.TwitchAccount static method*), 59  
`check_user_exists()` (*IreneAPIWrapper.models.TwitterAccount static method*), 57  
`city` (*IreneAPIWrapper.models.Location attribute*), 72  
`companies` (*IreneAPIWrapper.models.Preload attribute*), 69  
`Company` (*class in IreneAPIWrapper.models*), 46

company (*IreneAPIWrapper.models.Group* attribute), 37  
 connect() (*IreneAPIWrapper.models.IreneAPIClient* method), 2  
 connected (*IreneAPIWrapper.models.IreneAPIClient* attribute), 1  
 content (*IreneAPIWrapper.models.Tweet* attribute), 61  
 country (*IreneAPIWrapper.models.Location* attribute), 72  
 create() (*IreneAPIWrapper.models.AbstractModel* static method), 3  
 create() (*IreneAPIWrapper.models.Affiliation* static method), 31  
 create() (*IreneAPIWrapper.models.BloodType* static method), 62  
 create() (*IreneAPIWrapper.models.Channel* static method), 7  
 create() (*IreneAPIWrapper.models.Company* static method), 47  
 create() (*IreneAPIWrapper.models.Date* static method), 66  
 create() (*IreneAPIWrapper.models.Display* static method), 41  
 create() (*IreneAPIWrapper.models.EightBallResponse* static method), 18  
 create() (*IreneAPIWrapper.models.Fandom* static method), 40  
 create() (*IreneAPIWrapper.models.Group* static method), 38  
 create() (*IreneAPIWrapper.models.GroupAlias* static method), 50  
 create() (*IreneAPIWrapper.models.GuessingGame* static method), 14  
 create() (*IreneAPIWrapper.models.Guild* static method), 24  
 create() (*IreneAPIWrapper.models.Interaction* static method), 9  
 create() (*IreneAPIWrapper.models.Language* static method), 20  
 create() (*IreneAPIWrapper.models.Location* static method), 72  
 create() (*IreneAPIWrapper.models.Media* static method), 27  
 create() (*IreneAPIWrapper.models.Name* static method), 64  
 create() (*IreneAPIWrapper.models.Notification* static method), 11  
 create() (*IreneAPIWrapper.models.Person* static method), 35  
 create() (*IreneAPIWrapper.models.PersonAlias* static method), 49  
 create() (*IreneAPIWrapper.models.Position* static method), 45  
 create() (*IreneAPIWrapper.models.Social* static method), 44  
 create() (*IreneAPIWrapper.models.Tag* static method), 65  
 create() (*IreneAPIWrapper.models.TwitchAccount* static method), 59  
 create() (*IreneAPIWrapper.models.TwitterAccount* static method), 57  
 create() (*IreneAPIWrapper.models.UnscrambleGame* static method), 16  
 create() (*IreneAPIWrapper.models.User* static method), 29  
 create() (*IreneAPIWrapper.models.UserStatus* static method), 73  
 create\_bulk() (*IreneAPIWrapper.models.AbstractModel* static method), 3  
 create\_bulk() (*IreneAPIWrapper.models.TwitchAccount* static method), 60  
 create\_bulk() (*IreneAPIWrapper.models.TwitterAccount* static method), 57  
 create\_date (*IreneAPIWrapper.models.Guild* attribute), 23

## D

Date (class in *IreneAPIWrapper.models*), 66  
 date (*IreneAPIWrapper.models.Company* attribute), 47  
 date (*IreneAPIWrapper.models.Group* attribute), 37  
 date (*IreneAPIWrapper.models.Person* attribute), 34  
 date\_id (*IreneAPIWrapper.models.GuessingGame* attribute), 13  
 date\_id (*IreneAPIWrapper.models.UnscrambleGame* attribute), 15  
 dates (*IreneAPIWrapper.models.Preload* attribute), 70  
 delete() (*IreneAPIWrapper.models.AbstractModel* method), 3  
 delete() (*IreneAPIWrapper.models.Affiliation* method), 32  
 delete() (*IreneAPIWrapper.models.Channel* method), 7  
 delete() (*IreneAPIWrapper.models.Company* method), 47  
 delete() (*IreneAPIWrapper.models.Date* method), 67  
 delete() (*IreneAPIWrapper.models.Display* method), 41  
 delete() (*IreneAPIWrapper.models.EightBallResponse* method), 18  
 delete() (*IreneAPIWrapper.models.Fandom* method), 40  
 delete() (*IreneAPIWrapper.models.Group* method), 38  
 delete() (*IreneAPIWrapper.models.GroupAlias* method), 51  
 delete() (*IreneAPIWrapper.models.GuessingGame* method), 14  
 delete() (*IreneAPIWrapper.models.Guild* method), 24

`delete()` (*IreneAPIWrapper.models.Interaction method*), 9  
`delete()` (*IreneAPIWrapper.models.InteractionType method*), 10  
`delete()` (*IreneAPIWrapper.models.Location method*), 72  
`delete()` (*IreneAPIWrapper.models.Media method*), 27  
`delete()` (*IreneAPIWrapper.models.Name method*), 64  
`delete()` (*IreneAPIWrapper.models.Notification method*), 11  
`delete()` (*IreneAPIWrapper.models.Person method*), 35  
`delete()` (*IreneAPIWrapper.models.PersonAlias method*), 49  
`delete()` (*IreneAPIWrapper.models.Position method*), 45  
`delete()` (*IreneAPIWrapper.models.Social method*), 44  
`delete()` (*IreneAPIWrapper.models.Tag method*), 65  
`delete()` (*IreneAPIWrapper.models.TwitterAccount method*), 57  
`delete()` (*IreneAPIWrapper.models.UnscrambleGame method*), 16  
`delete()` (*IreneAPIWrapper.models.User method*), 29  
`delete()` (*IreneAPIWrapper.models.UserStatus method*), 73  
`delete_prefix()` (*IreneAPIWrapper.models.Guild method*), 24  
`delete_token()` (*IreneAPIWrapper.models.User method*), 29  
`description` (*IreneAPIWrapper.models.Company attribute*), 47  
`description` (*IreneAPIWrapper.models.Group attribute*), 37  
`description` (*IreneAPIWrapper.models.Guild attribute*), 22  
`description` (*IreneAPIWrapper.models.Person attribute*), 34  
`Difficulty` (*class in IreneAPIWrapper.models*), 71  
`difficulty` (*IreneAPIWrapper.models.GuessingGame attribute*), 13  
`difficulty` (*IreneAPIWrapper.models.Media property*), 27  
`difficulty` (*IreneAPIWrapper.models.UnscrambleGame attribute*), 16  
`disconnect()` (*IreneAPIWrapper.models.IreneAPIClient method*), 2  
`Display` (*class in IreneAPIWrapper.models*), 41  
`display` (*IreneAPIWrapper.models.Group attribute*), 37  
`display` (*IreneAPIWrapper.models.Person attribute*), 34  
`displays` (*IreneAPIWrapper.models.Preload attribute*), 69  
`done` (*IreneAPIWrapper.models.CallBack attribute*), 52  
`download_and_get_image_host_url()` (*IreneAPIWrapper.models.MediaSource method*), 4

## E

`eight_ball_responses` (*IreneAPIWrapper.models.Preload attribute*), 71  
`EightBallResponse` (*class in IreneAPIWrapper.models*), 17  
`emoji_count` (*IreneAPIWrapper.models.Guild attribute*), 22  
`emoji_limit` (*IreneAPIWrapper.models.Guild attribute*), 23  
`Empty`, 75  
`end` (*IreneAPIWrapper.models.Date attribute*), 66

## F

`facebook` (*IreneAPIWrapper.models.Social attribute*), 43  
`faces` (*IreneAPIWrapper.models.Media attribute*), 26  
`FailedObjectCreation`, 75  
`fancafe` (*IreneAPIWrapper.models.Social attribute*), 43  
`Fandom` (*class in IreneAPIWrapper.models*), 39  
`fandoms` (*IreneAPIWrapper.models.Preload attribute*), 70  
`fetch()` (*IreneAPIWrapper.models.AbstractModel static method*), 3  
`fetch()` (*IreneAPIWrapper.models.Affiliation static method*), 32  
`fetch()` (*IreneAPIWrapper.models.BloodType static method*), 62  
`fetch()` (*IreneAPIWrapper.models.Channel static method*), 7  
`fetch()` (*IreneAPIWrapper.models.Company static method*), 47  
`fetch()` (*IreneAPIWrapper.models.Date static method*), 67  
`fetch()` (*IreneAPIWrapper.models.Display static method*), 41  
`fetch()` (*IreneAPIWrapper.models.EightBallResponse static method*), 18  
`fetch()` (*IreneAPIWrapper.models.Fandom static method*), 40  
`fetch()` (*IreneAPIWrapper.models.Group static method*), 38  
`fetch()` (*IreneAPIWrapper.models.GroupAlias static method*), 51  
`fetch()` (*IreneAPIWrapper.models.GuessingGame static method*), 14  
`fetch()` (*IreneAPIWrapper.models.Guild static method*), 24  
`fetch()` (*IreneAPIWrapper.models.Location static method*), 72  
`fetch()` (*IreneAPIWrapper.models.Media static method*), 27  
`fetch()` (*IreneAPIWrapper.models.Name static method*), 64  
`fetch()` (*IreneAPIWrapper.models.Notification static method*), 11

`fetch()` (*IreneAPIWrapper.models.Person* static method), 35  
`fetch()` (*IreneAPIWrapper.models.PersonAlias* static method), 49  
`fetch()` (*IreneAPIWrapper.models.Position* static method), 45  
`fetch()` (*IreneAPIWrapper.models.Social* static method), 44  
`fetch()` (*IreneAPIWrapper.models.Tag* static method), 65  
`fetch()` (*IreneAPIWrapper.models.TwitchAccount* static method), 60  
`fetch()` (*IreneAPIWrapper.models.TwitterAccount* static method), 57  
`fetch()` (*IreneAPIWrapper.models.UnscrambleGame* static method), 16  
`fetch()` (*IreneAPIWrapper.models.User* static method), 29  
`fetch()` (*IreneAPIWrapper.models.UserStatus* static method), 74  
`fetch_all()` (*IreneAPIWrapper.models.AbstractModel* static method), 3  
`fetch_all()` (*IreneAPIWrapper.models.Affiliation* static method), 32  
`fetch_all()` (*IreneAPIWrapper.models.BloodType* static method), 63  
`fetch_all()` (*IreneAPIWrapper.models.Channel* static method), 8  
`fetch_all()` (*IreneAPIWrapper.models.Company* static method), 47  
`fetch_all()` (*IreneAPIWrapper.models.Date* static method), 67  
`fetch_all()` (*IreneAPIWrapper.models.Display* static method), 41  
`fetch_all()` (*IreneAPIWrapper.models.EightBallResponse* static method), 18  
`fetch_all()` (*IreneAPIWrapper.models.Fandom* static method), 40  
`fetch_all()` (*IreneAPIWrapper.models.Group* static method), 38  
`fetch_all()` (*IreneAPIWrapper.models.GroupAlias* static method), 51  
`fetch_all()` (*IreneAPIWrapper.models.GuessingGame* static method), 14  
`fetch_all()` (*IreneAPIWrapper.models.Guild* static method), 24  
`fetch_all()` (*IreneAPIWrapper.models.Interaction* static method), 9  
`fetch_all()` (*IreneAPIWrapper.models.Language* static method), 20  
`fetch_all()` (*IreneAPIWrapper.models.Location* static method), 72  
`fetch_all()` (*IreneAPIWrapper.models.Media* static method), 27  
`fetch_all()` (*IreneAPIWrapper.models.Name* static method), 64  
`fetch_all()` (*IreneAPIWrapper.models.Notification* static method), 11  
`fetch_all()` (*IreneAPIWrapper.models.Person* static method), 35  
`fetch_all()` (*IreneAPIWrapper.models.PersonAlias* static method), 49  
`fetch_all()` (*IreneAPIWrapper.models.Position* static method), 46  
`fetch_all()` (*IreneAPIWrapper.models.Social* static method), 44  
`fetch_all()` (*IreneAPIWrapper.models.Tag* static method), 65  
`fetch_all()` (*IreneAPIWrapper.models.TwitchAccount* static method), 60  
`fetch_all()` (*IreneAPIWrapper.models.TwitterAccount* static method), 57  
`fetch_all()` (*IreneAPIWrapper.models.UnscrambleGame* static method), 16  
`fetch_all()` (*IreneAPIWrapper.models.User* static method), 29  
`fetch_all()` (*IreneAPIWrapper.models.UserStatus* static method), 74  
`fetch_all_prefixes()` (*IreneAPIWrapper.models.Guild* static method), 24  
`fetch_prefixes()` (*IreneAPIWrapper.models.Guild* static method), 24  
`fetch_timeline()` (*IreneAPIWrapper.models.TwitterAccount* static method), 58  
`fetch_winners()` (*IreneAPIWrapper.models.BiasGame* static method), 12  
File (class in *IreneAPIWrapper.models*), 5  
file\_type (*IreneAPIWrapper.models.MediaSource* attribute), 4  
first (*IreneAPIWrapper.models.Name* attribute), 63  
force (*IreneAPIWrapper.models.Preload* attribute), 68  
former\_name (*IreneAPIWrapper.models.Person* attribute), 34

## G

gender (*IreneAPIWrapper.models.Person* attribute), 34  
generate\_bracket() (*IreneAPIWrapper.models.BiasGame* static method), 12  
generate\_pvp() (*IreneAPIWrapper.models.BiasGame* static method), 12  
get() (*IreneAPIWrapper.models.AbstractModel* static method), 3  
get() (*IreneAPIWrapper.models.Affiliation* static method), 32  
get() (*IreneAPIWrapper.models.BloodType* static method), 63



`get()` (*IreneAPIWrapper.models.Channel* static method), 8  
`get()` (*IreneAPIWrapper.models.Company* static method), 47  
`get()` (*IreneAPIWrapper.models.Date* static method), 67  
`get()` (*IreneAPIWrapper.models.Display* static method), 41  
`get()` (*IreneAPIWrapper.models.EightBallResponse* static method), 18  
`get()` (*IreneAPIWrapper.models.Fandom* static method), 40  
`get()` (*IreneAPIWrapper.models.Group* static method), 38  
`get()` (*IreneAPIWrapper.models.GroupAlias* static method), 51  
`get()` (*IreneAPIWrapper.models.GuessingGame* static method), 14  
`get()` (*IreneAPIWrapper.models.Guild* static method), 25  
`get()` (*IreneAPIWrapper.models.InteractionType* static method), 10  
`get()` (*IreneAPIWrapper.models.Location* static method), 72  
`get()` (*IreneAPIWrapper.models.Media* static method), 27  
`get()` (*IreneAPIWrapper.models.Name* static method), 64  
`get()` (*IreneAPIWrapper.models.Notification* static method), 11  
`get()` (*IreneAPIWrapper.models.PackMessage* method), 19  
`get()` (*IreneAPIWrapper.models.Person* static method), 35  
`get()` (*IreneAPIWrapper.models.PersonAlias* static method), 49  
`get()` (*IreneAPIWrapper.models.Position* static method), 46  
`get()` (*IreneAPIWrapper.models.Social* static method), 44  
`get()` (*IreneAPIWrapper.models.Tag* static method), 65  
`get()` (*IreneAPIWrapper.models.TwitchAccount* static method), 60  
`get()` (*IreneAPIWrapper.models.TwitterAccount* static method), 58  
`get()` (*IreneAPIWrapper.models.UnscrambleGame* static method), 16  
`get()` (*IreneAPIWrapper.models.User* static method), 29  
`get()` (*IreneAPIWrapper.models.UserStatus* static method), 74  
`get_all()` (*IreneAPIWrapper.models.Affiliation* static method), 32  
`get_all()` (*IreneAPIWrapper.models.BloodType* static method), 63  
`get_all()` (*IreneAPIWrapper.models.Channel* static method), 8  
`get_all()` (*IreneAPIWrapper.models.Company* static method), 47  
`get_all()` (*IreneAPIWrapper.models.Date* static method), 67  
`get_all()` (*IreneAPIWrapper.models.Display* static method), 42  
`get_all()` (*IreneAPIWrapper.models.EightBallResponse* static method), 18  
`get_all()` (*IreneAPIWrapper.models.Fandom* static method), 40  
`get_all()` (*IreneAPIWrapper.models.Group* static method), 39  
`get_all()` (*IreneAPIWrapper.models.GroupAlias* static method), 51  
`get_all()` (*IreneAPIWrapper.models.GuessingGame* static method), 14  
`get_all()` (*IreneAPIWrapper.models.Guild* static method), 25  
`get_all()` (*IreneAPIWrapper.models.Interaction* static method), 9  
`get_all()` (*IreneAPIWrapper.models.InteractionType* static method), 10  
`get_all()` (*IreneAPIWrapper.models.Location* static method), 73  
`get_all()` (*IreneAPIWrapper.models.Media* static method), 27  
`get_all()` (*IreneAPIWrapper.models.Name* static method), 64  
`get_all()` (*IreneAPIWrapper.models.Notification* static method), 12  
`get_all()` (*IreneAPIWrapper.models.Person* static method), 36  
`get_all()` (*IreneAPIWrapper.models.PersonAlias* static method), 49  
`get_all()` (*IreneAPIWrapper.models.Position* static method), 46  
`get_all()` (*IreneAPIWrapper.models.Social* static method), 44  
`get_all()` (*IreneAPIWrapper.models.Tag* static method), 66  
`get_all()` (*IreneAPIWrapper.models.TwitchAccount* static method), 60  
`get_all()` (*IreneAPIWrapper.models.TwitterAccount* static method), 58  
`get_all()` (*IreneAPIWrapper.models.UnscrambleGame* static method), 16  
`get_all()` (*IreneAPIWrapper.models.User* static method), 30  
`get_all()` (*IreneAPIWrapper.models.UserStatus* static method), 74  
`get_card()` (*IreneAPIWrapper.models.AbstractModel* method), 3

`get_card()` (*IreneAPIWrapper.models.Affiliation* method), 32  
`get_card()` (*IreneAPIWrapper.models.Company* method), 47  
`get_card()` (*IreneAPIWrapper.models.Date* method), 67  
`get_card()` (*IreneAPIWrapper.models.Display* method), 42  
`get_card()` (*IreneAPIWrapper.models.Group* method), 39  
`get_card()` (*IreneAPIWrapper.models.Name* method), 64  
`get_card()` (*IreneAPIWrapper.models.Person* method), 36  
`get_card()` (*IreneAPIWrapper.models.Social* method), 44  
`get_english()` (*IreneAPIWrapper.models.Language* static method), 20  
`get_input_count()` (*IreneAPIWrapper.models.PackMessage* static method), 19  
`get_lang()` (*IreneAPIWrapper.models.Language* static method), 20  
`get_lang_by_id()` (*IreneAPIWrapper.models.Language* static method), 20  
`get_posted()` (*IreneAPIWrapper.models.TwitchAccount* method), 60  
`get_random()` (*IreneAPIWrapper.models.Media* static method), 28  
`get_random_response()` (*IreneAPIWrapper.models.EightBallResponse* static method), 18  
`get_role_id()` (*IreneAPIWrapper.models.Subscription* method), 56  
`get_twitter_id()` (*IreneAPIWrapper.models.TwitterAccount* static method), 58  
`Group` (class in *IreneAPIWrapper.models*), 36  
`group` (*IreneAPIWrapper.models.Affiliation* attribute), 31  
`group_aliases` (*IreneAPIWrapper.models.Preload* attribute), 68  
`group_id` (*IreneAPIWrapper.models.GroupAlias* attribute), 50  
`GroupAlias` (class in *IreneAPIWrapper.models*), 50  
`groups` (*IreneAPIWrapper.models.Preload* attribute), 69  
`GuessingGame` (class in *IreneAPIWrapper.models*), 13  
`Guild` (class in *IreneAPIWrapper.models*), 21  
`guild_id` (*IreneAPIWrapper.models.Alias* attribute), 5  
`guild_id` (*IreneAPIWrapper.models.Channel* attribute), 7  
`guild_id` (*IreneAPIWrapper.models.GroupAlias* attribute), 50  
`guild_id` (*IreneAPIWrapper.models.Notification* attribute), 11  
`guild_id` (*IreneAPIWrapper.models.PersonAlias* attribute), 49  
`guilds` (*IreneAPIWrapper.models.Preload* attribute), 69

## H

`has_bot` (*IreneAPIWrapper.models.Guild* attribute), 23  
`height` (*IreneAPIWrapper.models.Person* attribute), 34

## I

`icon` (*IreneAPIWrapper.models.Guild* attribute), 22  
`id` (*IreneAPIWrapper.models.Access* attribute), 62  
`id` (*IreneAPIWrapper.models.Affiliation* attribute), 31  
`id` (*IreneAPIWrapper.models.Alias* attribute), 4  
`id` (*IreneAPIWrapper.models.BloodType* attribute), 62  
`id` (*IreneAPIWrapper.models.CallBack* attribute), 52  
`id` (*IreneAPIWrapper.models.Channel* attribute), 7  
`id` (*IreneAPIWrapper.models.Company* attribute), 46  
`id` (*IreneAPIWrapper.models.Date* attribute), 66  
`id` (*IreneAPIWrapper.models.Difficulty* attribute), 71  
`id` (*IreneAPIWrapper.models.Display* attribute), 41  
`id` (*IreneAPIWrapper.models.EightBallResponse* attribute), 17  
`id` (*IreneAPIWrapper.models.Fandom* attribute), 39  
`id` (*IreneAPIWrapper.models.Group* attribute), 37  
`id` (*IreneAPIWrapper.models.GroupAlias* attribute), 50  
`id` (*IreneAPIWrapper.models.Guild* attribute), 21  
`id` (*IreneAPIWrapper.models.Interaction* attribute), 8  
`id` (*IreneAPIWrapper.models.InteractionType* attribute), 9  
`id` (*IreneAPIWrapper.models.Location* attribute), 72  
`id` (*IreneAPIWrapper.models.Media* attribute), 26  
`id` (*IreneAPIWrapper.models.Mode* attribute), 68  
`id` (*IreneAPIWrapper.models.Name* attribute), 63  
`id` (*IreneAPIWrapper.models.Notification* attribute), 11  
`id` (*IreneAPIWrapper.models.Person* attribute), 33  
`id` (*IreneAPIWrapper.models.PersonAlias* attribute), 48  
`id` (*IreneAPIWrapper.models.Position* attribute), 45  
`id` (*IreneAPIWrapper.models.Social* attribute), 43  
`id` (*IreneAPIWrapper.models.Subscription* attribute), 55  
`id` (*IreneAPIWrapper.models.Tag* attribute), 65  
`id` (*IreneAPIWrapper.models.Tweet* attribute), 61  
`id` (*IreneAPIWrapper.models.TwitterAccount* attribute), 57  
`id` (*IreneAPIWrapper.models.UserStatus* attribute), 73  
`in_testing` (*IreneAPIWrapper.models.IreneAPIClient* attribute), 1  
`IncorrectNumberOfItems`, 75  
`insert()` (*IreneAPIWrapper.models.AbstractModel* static method), 3  
`insert()` (*IreneAPIWrapper.models.Affiliation* static method), 32  
`insert()` (*IreneAPIWrapper.models.Channel* static method), 8  
`insert()` (*IreneAPIWrapper.models.Company* static method), 48

`insert()` (*IreneAPIWrapper.models.Date* static method), 67  
`insert()` (*IreneAPIWrapper.models.Display* static method), 42  
`insert()` (*IreneAPIWrapper.models.EightBallResponse* static method), 19  
`insert()` (*IreneAPIWrapper.models.Fandom* static method), 40  
`insert()` (*IreneAPIWrapper.models.Group* static method), 39  
`insert()` (*IreneAPIWrapper.models.GroupAlias* static method), 51  
`insert()` (*IreneAPIWrapper.models.GuessingGame* static method), 15  
`insert()` (*IreneAPIWrapper.models.Guild* static method), 25  
`insert()` (*IreneAPIWrapper.models.Interaction* static method), 9  
`insert()` (*IreneAPIWrapper.models.InteractionType* static method), 10  
`insert()` (*IreneAPIWrapper.models.Location* static method), 73  
`insert()` (*IreneAPIWrapper.models.Media* static method), 28  
`insert()` (*IreneAPIWrapper.models.Name* static method), 64  
`insert()` (*IreneAPIWrapper.models.Notification* static method), 12  
`insert()` (*IreneAPIWrapper.models.Person* static method), 36  
`insert()` (*IreneAPIWrapper.models.PersonAlias* static method), 49  
`insert()` (*IreneAPIWrapper.models.Position* static method), 46  
`insert()` (*IreneAPIWrapper.models.Social* static method), 44  
`insert()` (*IreneAPIWrapper.models.Tag* static method), 66  
`insert()` (*IreneAPIWrapper.models.TwitchAccount* static method), 60  
`insert()` (*IreneAPIWrapper.models.TwitterAccount* static method), 58  
`insert()` (*IreneAPIWrapper.models.UnscrambleGame* static method), 17  
`insert()` (*IreneAPIWrapper.models.User* static method), 30  
`insert()` (*IreneAPIWrapper.models.UserStatus* static method), 74  
`instagram` (*IreneAPIWrapper.models.Social* attribute), 43  
`Interaction` (class in *IreneAPIWrapper.models*), 8  
`interactions` (*IreneAPIWrapper.models.Preload* attribute), 71  
`InteractionType` (class in *IreneAPIWrapper.models*), 9  
`internal_delete()` (in module *IreneAPIWrapper.models.base.receiver*), 5  
`internal_fetch()` (in module *IreneAPIWrapper.models.base.receiver*), 5  
`internal_fetch_all()` (in module *IreneAPIWrapper.models.base.receiver*), 6  
`internal_insert()` (in module *IreneAPIWrapper.models.base.receiver*), 6  
`InvalidToken`, 75  
`IreneAPIClient` (class in *IreneAPIWrapper.models*), 1  
`IreneAPIWrapper.models.base.receiver` module, 5  
`is_enabled` (*IreneAPIWrapper.models.Media* attribute), 26  
`is_nsfw` (*IreneAPIWrapper.models.GuessingGame* attribute), 14  
`is_nsfw` (*IreneAPIWrapper.models.Media* attribute), 26  
`is_preloaded` (*IreneAPIWrapper.models.IreneAPIClient* property), 2  
`is_reply` (*IreneAPIWrapper.models.Tweet* property), 61  
`is_retweet` (*IreneAPIWrapper.models.Tweet* property), 61  
**L**  
`label` (*IreneAPIWrapper.models.PackMessage* attribute), 19  
`Language` (class in *IreneAPIWrapper.models*), 20  
`language_id` (*IreneAPIWrapper.models.PackMessage* attribute), 19  
`languages` (*IreneAPIWrapper.models.Preload* attribute), 70  
`last` (*IreneAPIWrapper.models.Name* attribute), 63  
`latest_tweet` (*IreneAPIWrapper.models.Timeline* attribute), 55  
`latest_tweet` (*IreneAPIWrapper.models.Timeline* property), 55  
`latest_tweet` (*IreneAPIWrapper.models.TwitterAccount* attribute), 57  
`latest_tweet` (*IreneAPIWrapper.models.TwitterAccount* property), 58  
`Location` (class in *IreneAPIWrapper.models*), 72  
`location` (*IreneAPIWrapper.models.Person* attribute), 34  
`locations` (*IreneAPIWrapper.models.Preload* attribute), 70  
`logger` (*IreneAPIWrapper.models.IreneAPIClient* attribute), 2  
**M**  
`Media` (class in *IreneAPIWrapper.models*), 26  
`media` (*IreneAPIWrapper.models.Preload* attribute), 69  
`media_count` (*IreneAPIWrapper.models.Group* attribute), 38

media\_count (*IreneAPIWrapper.models.Person* attribute), 35  
media\_id (*IreneAPIWrapper.models.MediaSource* attribute), 4  
media\_ids (*IreneAPIWrapper.models.GuessingGame* attribute), 13  
MediaSource (class in *IreneAPIWrapper.models*), 4  
melon (*IreneAPIWrapper.models.Social* attribute), 43  
member\_count (*IreneAPIWrapper.models.Guild* attribute), 23  
message (*IreneAPIWrapper.models.PackMessage* attribute), 19  
mfa\_level (*IreneAPIWrapper.models.Guild* attribute), 22  
Mode (class in *IreneAPIWrapper.models*), 68  
mode\_id (*IreneAPIWrapper.models.GuessingGame* attribute), 13  
mode\_id (*IreneAPIWrapper.models.UnscrambleGame* attribute), 16  
module  
    *IreneAPIWrapper.models.base.receiver*, 5

## N

Name (class in *IreneAPIWrapper.models*), 63  
name (*IreneAPIWrapper.models.Alias* attribute), 4  
name (*IreneAPIWrapper.models.BloodType* attribute), 62  
name (*IreneAPIWrapper.models.Company* attribute), 46  
name (*IreneAPIWrapper.models.Difficulty* attribute), 71  
name (*IreneAPIWrapper.models.Fandom* attribute), 40  
name (*IreneAPIWrapper.models.Group* attribute), 37  
name (*IreneAPIWrapper.models.GroupAlias* attribute), 50  
name (*IreneAPIWrapper.models.Guild* attribute), 21  
name (*IreneAPIWrapper.models.InteractionType* attribute), 10  
name (*IreneAPIWrapper.models.Mode* attribute), 68  
name (*IreneAPIWrapper.models.Person* attribute), 34  
name (*IreneAPIWrapper.models.PersonAlias* attribute), 48  
name (*IreneAPIWrapper.models.Position* attribute), 45  
name (*IreneAPIWrapper.models.Subscription* attribute), 56  
name (*IreneAPIWrapper.models.Tag* attribute), 65  
name (*IreneAPIWrapper.models.TwitterAccount* attribute), 57  
names (*IreneAPIWrapper.models.Preload* attribute), 70  
nitro\_level (*IreneAPIWrapper.models.Guild* attribute), 22  
Notification (class in *IreneAPIWrapper.models*), 10  
notifications (*IreneAPIWrapper.models.Preload* attribute), 71  
num\_inputs (*IreneAPIWrapper.models.PackMessage* attribute), 19

## O

origin (*IreneAPIWrapper.models.IreneAPIClient* attribute), 2  
owner (*IreneAPIWrapper.models.Guild* attribute), 22  
owner\_id (*IreneAPIWrapper.models.Guild* attribute), 22

## P

PackMessage (class in *IreneAPIWrapper.models*), 19  
Person (class in *IreneAPIWrapper.models*), 33  
person (*IreneAPIWrapper.models.Affiliation* attribute), 31  
person\_aliases (*IreneAPIWrapper.models.Preload* attribute), 68  
person\_id (*IreneAPIWrapper.models.PersonAlias* attribute), 48  
PersonAlias (class in *IreneAPIWrapper.models*), 48  
persons (*IreneAPIWrapper.models.Preload* attribute), 69  
phrase (*IreneAPIWrapper.models.Notification* attribute), 11  
Position (class in *IreneAPIWrapper.models*), 45  
positions (*IreneAPIWrapper.models.Affiliation* attribute), 31  
positions (*IreneAPIWrapper.models.Preload* attribute), 70  
prefixes (*IreneAPIWrapper.models.Guild* attribute), 24  
Preload (class in *IreneAPIWrapper.models*), 68

## Q

query() (*IreneAPIWrapper.models.Wolfram* static method), 17

## R

reaction\_role\_messages (*IreneAPIWrapper.models.Preload* attribute), 71  
reconnect (*IreneAPIWrapper.models.IreneAPIClient* attribute), 2  
request (*IreneAPIWrapper.models.CallBack* attribute), 52  
response (*IreneAPIWrapper.models.CallBack* attribute), 52  
response (*IreneAPIWrapper.models.EightBallResponse* attribute), 17  
role\_count (*IreneAPIWrapper.models.Guild* attribute), 23

## S

score (*IreneAPIWrapper.models.UserStatus* attribute), 73  
set\_as\_done() (*IreneAPIWrapper.models.CallBack* method), 52  
set\_ban() (*IreneAPIWrapper.models.User* method), 30



- set\_data\_mod() (*IreneAPIWrapper.models.User method*), 30  
 set\_mod() (*IreneAPIWrapper.models.User method*), 30  
 set\_patron() (*IreneAPIWrapper.models.User method*), 30  
 set\_proofreader() (*IreneAPIWrapper.models.User method*), 30  
 set\_super\_patron() (*IreneAPIWrapper.models.User method*), 30  
 set\_translator() (*IreneAPIWrapper.models.User method*), 30  
 shard\_id (*IreneAPIWrapper.models.Guild attribute*), 23  
 short\_name (*IreneAPIWrapper.models.Language attribute*), 20  
 Social (*class in IreneAPIWrapper.models*), 42  
 social (*IreneAPIWrapper.models.Group attribute*), 37  
 social (*IreneAPIWrapper.models.Person attribute*), 34  
 socials (*IreneAPIWrapper.models.Preload attribute*), 70  
 source (*IreneAPIWrapper.models.Media attribute*), 26  
 splash (*IreneAPIWrapper.models.Guild attribute*), 22  
 spotify (*IreneAPIWrapper.models.Social attribute*), 43  
 stage\_name (*IreneAPIWrapper.models.Affiliation attribute*), 31  
 start (*IreneAPIWrapper.models.Date attribute*), 66  
 status\_ids (*IreneAPIWrapper.models.GuessingGame attribute*), 13  
 status\_ids (*IreneAPIWrapper.models.UnscrambleGame attribute*), 15  
 subbed\_in() (*IreneAPIWrapper.models.TwitchAccount static method*), 61  
 subbed\_in() (*IreneAPIWrapper.models.TwitterAccount static method*), 58  
 subscribe() (*IreneAPIWrapper.models.Subscription method*), 56  
 subscribe() (*IreneAPIWrapper.models.TwitchAccount method*), 61  
 subscribe() (*IreneAPIWrapper.models.TwitterAccount method*), 58  
 Subscription (*class in IreneAPIWrapper.models*), 55
- ## T
- Tag (*class in IreneAPIWrapper.models*), 65  
 tags (*IreneAPIWrapper.models.Group attribute*), 38  
 tags (*IreneAPIWrapper.models.Person attribute*), 35  
 tags (*IreneAPIWrapper.models.Preload attribute*), 68  
 text\_channel\_count (*IreneAPIWrapper.models.Guild attribute*), 23  
 tiktok (*IreneAPIWrapper.models.Social attribute*), 43  
 Timeline (*class in IreneAPIWrapper.models*), 55  
 token (*IreneAPIWrapper.models.IreneAPIClient attribute*), 1  
 Tweet (*class in IreneAPIWrapper.models*), 61  
 tweets (*IreneAPIWrapper.models.Timeline attribute*), 55  
 twitch\_subscriptions (*IreneAPIWrapper.models.Preload attribute*), 70  
 TwitchAccount (*class in IreneAPIWrapper.models*), 59  
 twitter (*IreneAPIWrapper.models.Social attribute*), 43  
 twitter\_accounts (*IreneAPIWrapper.models.Preload attribute*), 69  
 twitter\_subscriptions (*IreneAPIWrapper.models.Preload attribute*), 70  
 TwitterAccount (*class in IreneAPIWrapper.models*), 56  
 type (*IreneAPIWrapper.models.CallBack attribute*), 52  
 type (*IreneAPIWrapper.models.Interaction attribute*), 8
- ## U
- UnscrambleGame (*class in IreneAPIWrapper.models*), 15  
 unsubscribe() (*IreneAPIWrapper.models.Subscription method*), 56  
 unsubscribe() (*IreneAPIWrapper.models.TwitchAccount method*), 61  
 unsubscribe() (*IreneAPIWrapper.models.TwitterAccount method*), 59  
 update\_end\_date() (*IreneAPIWrapper.models.Date method*), 68  
 update\_media\_and\_status() (*IreneAPIWrapper.models.GuessingGame method*), 15  
 update\_posted() (*IreneAPIWrapper.models.TwitchAccount method*), 61  
 update\_score() (*IreneAPIWrapper.models.UserStatus method*), 74  
 update\_status() (*IreneAPIWrapper.models.UnscrambleGame method*), 17  
 update\_tweets() (*IreneAPIWrapper.models.Timeline method*), 55  
 upsert\_filter\_groups() (*IreneAPIWrapper.models.User method*), 30  
 upsert\_filter\_persons() (*IreneAPIWrapper.models.User method*), 31  
 upsert\_guesses() (*IreneAPIWrapper.models.Media method*), 28  
 upsert\_win() (*IreneAPIWrapper.models.BiasGame static method*), 13  
 url (*IreneAPIWrapper.models.Interaction attribute*), 9  
 url (*IreneAPIWrapper.models.Media property*), 28  
 url (*IreneAPIWrapper.models.MediaSource attribute*), 4  
 User (*class in IreneAPIWrapper.models*), 29  
 user\_id (*IreneAPIWrapper.models.IreneAPIClient attribute*), 1  
 user\_id (*IreneAPIWrapper.models.Notification attribute*), 11  
 user\_id (*IreneAPIWrapper.models.UserStatus attribute*), 73  
 users (*IreneAPIWrapper.models.Preload attribute*), 69  
 UserStatus (*class in IreneAPIWrapper.models*), 73

### V

`verbose` (*IreneAPIWrapper.models.IreneAPIClient* attribute), [2](#)

`vlive` (*IreneAPIWrapper.models.Social* attribute), [43](#)

`voice_channel_count` (*IreneAPIWrapper.models.Guild* attribute), [23](#)

### W

`wait_for_completion()` (*IreneAPIWrapper.models.CallBack* method), [53](#)

`website` (*IreneAPIWrapper.models.Group* attribute), [37](#)

`Wolfram` (class in *IreneAPIWrapper.models*), [17](#)

### Y

`youtube` (*IreneAPIWrapper.models.Social* attribute), [43](#)