IreneUtility

Release V1.04.0

MujyKun

Jun 29, 2021

CONTENTS:

CHAPTER

ONE

UTILITY CLIENT

CHAPTER

TWO

BASE

class `IreneUtility.Base.`**`Base`**(*utility_object*)

Base Class that will hold the utility object for a subclass. Meant to be a parent class.

CHAPTER

THREE

MODELS

3.1 __init__ (models)

class `IreneUtility.models.`**`BaseUtil`**

Serves as an extension to avoid Circular imports while maintaining the main Utility object across the models that need to utilize certain methods.

Note that we have this as a class so we do not need to constantly import if ex is None. This object makes it much easier to maintain.

base_util - The primary instance of *`BaseUtil`* that contains the primary `IreneUtility.Utility.Utility` Instance.

3.2 Album

class `IreneUtility.models.`**`Album`**(*album_name*, *idol_cards*, *rap_score=0*, *dance_score=0*, *vocal_score=0*, *popularity=0*, *income_rate=0*, *album_active_time=None*)

async calculate_dance_score()

Returns the total dance skills of all idols in the album

async calculate_rap_score()

Returns the total rap skills of all idols in the album

async calculate_vocal_score()

returns the total vocal skills of all idols in the album

async skill_completion_multiplier()

Provides a small bonus for having idols that fulfill all of the different categories.

3.3 BiasGame

class `IreneUtility.models.`**`BiasGame`**(**args*, *bracket_size=8*, *gender='all'*)

async check_message(*message*, *first_idol*, *second_idol*)

Check the reactions of the message and process results

async end_game()

End the game

async generate_brackets()
Generates the brackets and the idols going against each other

async process_game()
Process bias guessing game by sending messages and new questions until the game should end.

async run_current_bracket()
Generate a new question for the bias game.

3.4 BlackJackGame

class IreneUtility.models.**BlackJackGame**(**args*, *first_player*, *first_player_bet*)
BlackJack Game for two users.

async announce_winner()
Announce the winner of the game.

async calculate_score(*cards: List[*IreneUtility.models.playingcard.PlayingCard*]*) → int
Calculate the score of a player.

Parameters **cards** – List of PlayingCards the user has in their deck.

Returns Score of the player

async check_message()
Check incoming messages in the text channel and determines if the player wants to hit or stand.

async check_standing(*first_player=True*)
Check if a player is standing.

Parameters **first_player** – True if it is the first player that wants to stand. Otherwise its the second player.

Returns True if the user is standing.

async choose_random_card() → *IreneUtility.models.playingcard.PlayingCard*
Chooses a random card that is available in the deck.

async deal_with_bets()
Properly deal with bets and appropriately remove/add the bets from the players balances.

async determine_winner() → *IreneUtility.models.user.User*
Determine the winner of the blackjack game.

async end_game()
End the blackjack game.

async finalize_game()
Finalize the game by dealing with bets, announcing the winner, and officially ending the game.

async hit(*first_player=True*)
Let a player hit

Parameters **first_player** – True if it is the first player that wants to hit. Otherwise its the second player.

async process_game()
Start the blackjack game.

async stand(*first_player=True*)
Let a player stand

Parameters **first_player** – True if it is the first player that wants to stand. Otherwise its the second player.

3.5 File

class IreneUtility.models.**File**(*file_location*, *image_url*)

Represents an OS file.

async send_file(*channel: discord.channel.TextChannel*, *message=None*, *url=False*)

Parameters

- **channel** – Discord Text Channel to send the file to.
- **message** – Message followed with the image.
- **url** – True/False if the url should be posted instead.

3.6 GachaValues

class IreneUtility.models.**GachaValues**

3.7 Game

class IreneUtility.models.**Game**(*utility_obj*, *ctx*)

async end_game()

Ends a guessing game.

async process_game()

Starts processing the game.

3.8 Group

class IreneUtility.models.**Group**(***kwargs*)

A group of idols/celebrities

3.9 GuessingGame

class IreneUtility.models.**GuessingGame**(**args*, *max_rounds=20*, *timeout=20*, *gender='all'*, *difficulty='medium'*, *game_mode='idol'*)

async check_message()

Check incoming messages in the text channel and determine if it is correct.

async create_acceptable_answers()

Create acceptable answers.

async create_idol_pool()
Create the game's idol pool.

async create_new_question()
Create a new question and send it to the channel.

async credit_user(*user_id*)
Increment a user's score

async display_winners()
Displays the winners and their scores.

async end_game()
Ends a guessing game.

async print_answer(*question_skipped=False*, *dead_link=False*)
Prints the current round's answer.

async process_game()
Ignores errors and continuously makes new questions until the game should end.

async update_scores()
Updates all player scores

3.10 Idol

class IreneUtility.models.**Idol**(***kwargs*)
Represents an Idol/Celebrity.

3.11 IdolCard

class IreneUtility.models.**IdolCard**(*idol*, *card_owner: discord.user.User*, *issue_number=1*, *rap_skill=0*, *vocal_skill=0*, *dance_skill=0*, *rarity='common'*)

3.12 Keys

class IreneUtility.models.**Keys**(***kwargs*)

ACCESS_SECRET: str
Spotify

X_RapidAPI_headers: dict
Tenor

api_port: str
Bot Site

client: commands.AutoShardedBot
Reactions/Emojis Turned into Unicode Strings

client_session: AioHTTPClient
Wolfram

async connect_to_db()
Create a pool to the postgres database using asyncpg

datadog_app_key: str
BlackJack

db_conn: asyncpg.pool.Pool
Papago/Translator

discord_boats: discord_boats_client
Database Connection

get_kwarg(*kwarg_name*)
Get a kwarg

idol_photo_location: str
Twitch API

kwargs
Bot Tokens

last_fm_headers: dict
Patreon

lyric_client: lyrics_client
//github.com/MujyKun/Weverse

Type Weverse - https

next_emoji
Twitter

oxford_app_key: str
Urban Dictionary

patreon_super_role_id: int
AioHTTP

playing_card_location: str
Bot API

spotify_client_secret: str
Oxford

tenor_key: str
Top.gg

test_client_token: str
General

top_gg: DBLClient
Discord Boats

translate_private_key
LastFM

twitch_client_secret: str
//github.com/DataDog/datadogpy

Type DataDog - https

weverse_image_folder: str
GroupMembers Directories

wolfram_app_id: str
//github.com/KSoft-Si/ksoftapi.py

Type Lyrics API - https

3.13 PlayingCard

class IreneUtility.models.**PlayingCard**(**args*, *card_id*, *card_name*, *value*)
Represents a custom playing card.

3.14 User

class IreneUtility.models.**User**(*user_id: int*)
Represents a discord user.

async ensure_level()
Ensure the user has a row in the levels table.

async get_daily_amount()
Get the amount the user should receive daily.

async static get_needed_for_level(*level: int*, *column_name: str*)
Returns money/experience needed for a certain level.

async get_profile_xp()
Get the user's profile xp.

async get_rob_amount(*money*)
The amount to rob a specific person based on their rob level.

Parameters **money** – (The amount of money the person getting robbed has)

async get_rob_percentage()
Get the percentage of being able to rob. (Every 1 is 5%)

async get_shortened_balance() → str
Shorten an amount of money to its value places.

async register_currency()
Registers the user to the currency system.

async set_language(*language*)
Sets the user's language.

async set_level(*level*, *command_name*)
Set the level of a command/feature.

async set_profile_xp(*xp_amount*)
Set the user's profile xp.

async try_to_rob_user(*user*) → bool
Attempt to rob a user.

Parameters **user** – User to rob

Returns True if the user successfully robbed.

async update_balance(*balance: Optional[int] = None*, *add: Optional[int] = None*, *remove: Optional[int] = None*)

Set balance of user in db and object.

Parameters

- **balance** – The amount to set the balance to.
- **add** – The amount to add to the current balance.
- **remove** – The amount to remove from the current balance.

async update_level_in_db(*column_name*, *level*)

Update the level for the user.

CHAPTER

FOUR

UTIL

4.1 u_biasgame

class IreneUtility.util.u_biasgame.**BiasGame**(**args*)

async create_bias_game_image(*first_idol_id*, *second_idol_id*)
Uses thread pool to create bias game image to prevent IO blocking.

merge_images(*first_idol_id*, *second_idol_id*)
Merge Idol Images if the merge doesn't exist already.

4.2 u_blackjack

class IreneUtility.util.u_blackjack.**BlackJack**(**args*)

async find_game(*user*) → *IreneUtility.models.blackjackgame.BlackJackGame*
Find a blackjack game that a user is in.

Parameters **user** – A Utility User object, Context, or User ID

Returns BlackJack Game

async generate_playing_cards()
Generate custom playing cards with the background as an idol avatar.

merge_images(*card_file_name*, *idol_file_name*, *unique_id*)
Merges a template card with an idol avatar.

Parameters

- **card_file_name** – A Card's File name & type without the directory.
- **idol_file_name** – An Idol's File name & type without the directory.
- **unique_id** – The unique row id in the database table that will be the merged file name.

remove_all_card_files()
Remove all card files from OS.

4.3 u_cache

class IreneUtility.util.u_cache.**Cache**(**args*)

static apply_bold_to_braces(*text: str*) → str
Applys bold markdown in between braces.

async create_bot_bans()
Create the cache for banned users from the bot.

async create_bot_command_cache()
Create custom command cache

async create_cache(*on_boot_up=True*)
Create the general cache on startup

async create_command_counter()
Updates Cache for command counter and sessions

async create_currency_cache()
Create cache for currency

async create_dead_link_cache()
Creates Dead Link Cache

async create_disabled_games_cache()
Creates a list of channels with disabled games.

async create_gg_filter_cache()
Create filtering of guessing game cache.

async create_group_cache()
Create Group Objects and store them as cache

async create_groups()
Set cache for group photo count

async create_guessing_game_cache()
Create cache for guessing game scores

async create_guild_cache()
Update the DB Guild Cache. Useful for updating info for API.

async create_idol_cache()
Create Idol Objects and store them as cache.

async create_idols()
Set cache for idol photo count

async create_image_cache()
Creates Image objects and stores them in local cache.

Note that usage of these images is unnecessary as a call to the API would be more efficient. Therefore, IreneBot will not be using the image objects directly.

async create_language_cache()
Create cache for user languages.

async create_levels_cache()
Create the cache for user levels.

async create_logging_channels()
Create the cache for logged servers and channels.

async create_mod_mail()
Create the cache for existing mod mail

async create_n_word_counter()
Update NWord Cache

async create_patreons()
Create the cache for Patrons.

async create_playing_cards()
Crache cache for playing cards.

async create_reminder_cache()
Create cache for reminders

async create_restricted_channel_cache()
Create restricted idol channel cache

async create_self_assignable_role_cache()
Create cache for self assignable roles

async create_send_idol_photo_cache()
Creates the list of idols that needs to be sent to a text channel after t time.

async create_server_prefixes()
Create the cache for server prefixes.

async create_temp_channels()
Create the cache for temp channels.

async create_timezone_cache()
Create cache for timezones

async create_twitch_cache()
Create cache for twitch followings

async create_unscramble_game_cache()
Create cache for unscramble game scores

async create_user_notifications()
Set the cache for user phrases

async create_welcome_message_cache()
Create the cache for welcome messages.

async create_weverse_channel_cache()
Create cache for channels that are following a community on weverse.

async get_session_id()
Force get the session id, this will also set total used and the session id.

async load_language_packs()
Create cache for language packs.

async process_cache_time(*method*, *name*, **args*, ***kwargs*)
Process the cache time.

async process_session()
Sets the new session id, total used, and time format for distinguishing days.

async request_support_server_members()
Request the support server to be chunked and be put in cache.

We need this so that we can accurately determine if a user is in the support server or not.

async request_twitter_channel()
Fetch twitter channel and store it in cache.

4.4 u_customcommands

class IreneUtility.util.u_customcommands.**CustomCommands**(**args*)

4.5 u_database

class IreneUtility.util.u_database.**DataBase**(**args*)

async get_db_connection()
Retrieve Database Connection

4.6 u_datadog

class IreneUtility.util.u_datadog.**DataDog**(**args*)

get_metric_info()
Retrieves metric info to send to datadog.

initialize_data_dog()
Initialize The DataDog Class for metrics.

send_metric(*metric_name*, *value*)
Send a metric value to DataDog.

send_metrics()
Send metrics for all stats.

4.7 u_exceptions

exception IreneUtility.util.u_exceptions.**ImproperFormat**
Invalid Format was given.

exception IreneUtility.util.u_exceptions.**InvalidParamsPassed**(*msg*)
Raised when IDs are invalid for an add/remove/set method.

exception IreneUtility.util.u_exceptions.**Limit**
A limit was reached.

exception IreneUtility.util.u_exceptions.**MaxAttempts**(*msg*)
Essentially StopIteration, but created for logging to file & console upon raising error.

exception IreneUtility.util.u_exceptions.**NoKeyFound**(*msg*)
Raised when no keys were found.

exception IreneUtility.util.u_exceptions.**NoTimeZone**
No Timezone was found.

exception IreneUtility.util.u_exceptions.**Pass**
A hack exception. This exception was meant to occur in order to jump to a part in code. Indicates something went right instead of wrong.

exception IreneUtility.util.u_exceptions.**ShouldNotBeHere**(*msg*)
Raised when safe-guarded code is created. If this exception is raised, the code reached a point it should not have

exception IreneUtility.util.u_exceptions.**TooLarge**
The input was too long.

4.8 u_gacha

class IreneUtility.util.u_gacha.**Gacha**(**args*)

async static get_all_skill_scores(*idol_skill_type*, *card_rarity*)
Returns the rap, dance, and vocal scores of an idol

async static random_album_popularity()
Returns a truncated normal random popularity between 0 and 1 that follows the PDF f(y) = exp(-a * (y - c)^2) where a is the curvature and c is the bell center. This is a truncated normal distribution. The random variable transformation g(x) : x -> y needs to be used where x is a uniform distribution and y is the f(x) distribution. g(x) = Fy^-1(F(x)) where Fx = x and Fy = erf(sqrt(a)*(y - c)) which are the corresponding CDFs of x and y. Solving we find that g(x) = erfinv(x) / sqrt(a) + c.

async random_skill_score(*card_rarity*)
Return a random skill score for rap/dance/vocal for the gacha card between 1 and 99 dependent on the rarity of the card.

4.9 u_groupmembers

class IreneUtility.util.u_groupmembers.**GroupMembers**(**args*)

async check_channel_sending_photos(*channel_id*)
Checks a text channel ID to see if it is restricted from having idol photos sent.

async check_group_and_idol(*message_content*, *server_id=None*)
returns specific idols being called from a reference to a group ex: redvelvet irene

async check_idol_post_reactions(*message*, *user_msg*, *idol*, *link*, *guessing_game=False*)
Check the reactions on an idol post or guessing game.

async check_server_sending_photos(*server_id*)
Checks a server to see if it has a specific channel to send idol photos to

async static check_to_add_alias_to_list(*alias*, *name*, *mode=0*)
Check whether to add an alias to a list. Compares a name with an existing alias.

async choose_random_member(*members=None*, *groups=None*)
Choose a random member object from a member or group list given.

async delete_channel_from_send_idol(*text_channel*)
Deletes a channel permanently from the send idol cache.

Parameters **text_channel** – (discord.TextChannel or int) Key to pop from the cache. The client should know which it is.

async delete_restricted_channel_from_cache(*channel_id*, *send_all*)
Deletes restricted channel from cache.

async static format_card_fields(*obj*, *card_formats*)
Formats all relevant card fields to be displayed

async static get_all_groups()
Get all groups.

async get_all_images_count()
Get the amount of images the bot has.

async get_channel_sending_photos(*server_id*)
Returns a text channel from a server that requires idol photos to be sent to a specific text channel.

async get_db_aliases(*object_id*, *group=False*)
Get the aliases of an idol or group from the database.

async get_db_groups_from_member(*member_id*)
Return all the group ids an idol is in from the database.

async get_db_idol_called(*member_id*)
Get the amount of times an idol has been called from the database.

async get_db_members_in_group(*group_id*)
Get the members in a specific group from the database.

async get_google_drive_link(*api_url*)
Get the google drive link based on the api's image url.

async get_group(*group_id*) → *IreneUtility.models.group.Group*
Get a group by the group id.

async get_group_names_as_string(*idol*)
Get the group names split by a | .

async get_group_where_group_matches_name(*name*, *mode=0*, *server_id=None*)
Get group ids for a specific name.

async get_idol_by_image_id(*image_id*)
Get an idol object by the unique image id.

Returns Idol Object or NoneType

async get_idol_post_embed(*group_id*, *idol*, *photo_link*, *user_id=None*, *guild_id=None*, *guessing_game=False*, *scores=None*)
The embed for an idol post.

async get_idol_where_member_matches_name(*name*, *mode=0*, *server_id=None*)
Get idol object if the name matches an idol

async get_member(*idol_id*) → *IreneUtility.models.idol.Idol*
Get a member by the idol id.

async get_member_names_as_string(*group*)
Get the member names split by a | .

async get_random_idol()
Get a random idol with at least 1 photo.

async idol_post(*channel*, *idol*, ***kwargs*)
The main process for managing the errors behind an api call for an idol's photo.

Parameters

- **channel** – (discord.Channel) Channel that the embed/image should be posted to.
- **idol** – (IreneUtility Idol object) Idol that will be posted.
- **photo_link** – Image that will be embedded.
- **group_id** – Group ID the idol may be posted with. (used for if a group photo was called instead).
- **special_message** – Any message that should be applied to the post.
- **user_id** – User ID that is calling the photo.
- **guessing_game** – (bool) Whether the method is called from a guessing game.
- **scores** – (dict) Any scores that may come with a game. In a format of {user id : score}
- **msg_timeout** – Amount of time before deleting a message.

log_idol_command(*message*)
Log an idol photo that was called.

async manage_send_idol_photo(*text_channel*, *idol_id*, *limit=None*)
Adds/Removes/Updates idol ids based on the text channel that will be used to send idol photos after t time.

Parameters

- **text_channel** – discord.TextChannel or text channel id for the idol photo to be sent to
- **idol_id** – idol id to add or remove.
- **limit** – (int) the limit for what the text channel can have. If this is exceeded, u_exceptions.Limit will be raised.

Returns False if text channel input was incorrect. 'insert' if the idol id was inserted. 'remove' if the idol id was removed. 'delete' if the channel was completely removed from the table.

async process_names(*ctx*, *page_number_or_group*, *mode*)
Structures the input for idol names commands and sends information to transfer the names to the channels.

async remove_global_alias(*obj*, *alias*)
Remove a global idol/group alias

async remove_local_alias(*obj*, *alias*, *server_id*)
Remove a server idol/group alias

async request_image_post(*message*, *idol*, *channel*)
Checks if the user can post an image, then posts it.

async send_names(*ctx*, *mode*, *user_page_number=1*, *group_ids=None*)
Send the names of all idols in an embed with many pages.

async send_vote_message(*message*)
Send the vote message to a user.

async set_as_group_photo(*link*)
Set a photo as a group photo.

async set_embed_card_info(*obj*, *group=False*, *server_id=None*)
Sets General Information about a Group or Idol.

async set_embed_with_aliases(*name*, *server_id=None*)
Create an embed with the aliases of the names of groups or idols sent in

async set_embed_with_all_aliases(*mode*, *server_id=None*)
Send the names of all aliases in an embed with many pages.

async set_global_alias(*obj*, *alias*)
Set an idol/group alias for the bot.

async set_local_alias(*obj*, *alias*, *server_id*)
Set an idol/group alias for a server

async update_member_count(*idol*)
Update the amount of times an idol has been called.

4.10 u_guessinggame

class IreneUtility.util.u_guessinggame.**GuessingGame**(**args*)

async create_user_in_guessing_game(*user_id*)
Inserts a user into the guessing game db with no scores. This allows for updating scores easier.

async filter_add_group(*user*, *group*)
Adds a filtered group to a user.

async filter_auto_add_remove_group(*user_or_id*, *group_or_id*)
Automatically Add/Remove a group from a user's filtered group list based on the current list.

:returns False if group was removed. :returns True if group was added. :exception self.ex.exceptions.InvalidParamsPassed if invalid group id.

async filter_remove_group(*user*, *group*)
Remove a filtered group from a user.

async get_guessing_game_top_ten(*difficulty*, *members=None*)
Get the top ten of a certain guessing game difficulty

async toggle_filter(*user_id*)
Enables/Disables the group filter for the guessing game on a user.

async update_user_guessing_game_score(*difficulty*, *user_id*, *score*)
Update a user's guessing game score.

4.11 u_lastfm

class IreneUtility.util.u_lastfm.**LastFM**(**args*)

create_fm_payload(*method*, *user=None*, *limit=None*, *time_period=None*)
Creates the payload to be sent to Last FM

async get_fm_response(*method*, *user=None*, *limit=None*, *time_period=None*)
Receives the response from Last FM

async get_fm_username(*user_id*)
Gets Last FM username from the DB.

async set_fm_username(*user_id*, *username*)
Sets Last FM username to the DB.

4.12 u_local_cache

class IreneUtility.util.u_local_cache.**Cache**(**args*)

commands_used
{ server_id : {

send_all: 0 or 1, logging_channel: channel_id, channels: [channel_id, channel_id] },

… }

list_of_logged_channels
Welcome Messages { server_id : {

channel_id: channel_id, message: text, enabled: 0 or 1 }

}

server_prefixes
reset timer for idol photos (keeps track of command usage) {

reset_time: date userid: [commands_used, time_since_last_command]

}

twitch_channels_is_live
User Notifications and Mod Mail are constantly iterated over, therefore we need a synced list apart from the information present in the user objects to stop the bot from being blocked/behind on future commands. These were removed and put into the user objects, but will now be placed back due to this issue.

welcome_messages
Temp Channels { channel_id : seconds }

4.13 u_logger

IreneUtility.util.u_logger.**console**(*message*, *method=None*, *event_loop=None*)
Prints message to console and adds to logging.

Parameters

- **message** – The message that will be printed out and logged.
- **method** – The function/method that called this function.
- **event_loop** – An existing event loop.

IreneUtility.util.u_logger.**get_class**(*method*)
Returns the class that belongs to the method. :param method: The method that needs to be checked.

REF -> https://stackoverflow.com/a/25959545/13159093

IreneUtility.util.u_logger.**logfile**(*message*)
Logs a message to the info file.

Parameters **message** – The message that will be logged.

`IreneUtility.util.u_logger.manage_log(body_msg, log_type, method=None, event_loop=None)`
Process the type of logging it is and writes to file.

Parameters

- **body_msg** – (str) Line that should be appended to the file.
- **log_type** – (str) The end of the file name that differentiates the type of logging it is.
- **method** – The function/method that called this function.
- **event_loop** – An existing event loop.

`IreneUtility.util.u_logger.useless(message, method=None)`
Logs Try-Except-Passes. This will put the exceptions into a log file specifically for cases with no exception needed.

Parameters

- **message** – The message that will be logged.
- **method** – The function/method that called this function.

`async IreneUtility.util.u_logger.write_to_file(location, body_msg)`
Write a line to a file.

Parameters

- **location** – (str) File Location
- **body_msg** – (str) Line that should be appended to the file.

4.14 u_miscellaneous

`class IreneUtility.util.u_miscellaneous.Miscellaneous(*args)`

`async add_command_count(command_name)`
Add 1 to the specific command count and to the count of the current minute.

`async add_session_count()`
Adds one to the current session count for commands used and for the total used.

`async ban_user_from_bot(user_id)`
Bans a user from using the bot.

`async check_for_bot_mentions(message)`
Returns true if the message is only a bot mention and nothing else.

`async check_for_nword(message)`
Processes new messages that contains the N word.

`async check_if_bot_banned(user_id)`
Check if the user can use the bot.

`async static check_if_moderator(ctx)`
Check if a user is a moderator on a server

`async check_if_temp_channel(channel_id)`
Check if a channel is a temp channel

async check_message_is_command(*message*, *is_command_name=False*)
Check if a message is a command.

static check_message_not_empty(*message*)
Check if a message has content.

check_nword(*message_content*)
Check if a message contains the NWord.

async delete_temp_messages(*message*)
Delete messages that are temp channels

async disable_interaction(*server_id*, *interaction*)
Disable an interaction (to a specific server)

async enable_interaction(*server_id*, *interaction*)
Reenable an interaction that was disabled by a server

get_channel_count()
Returns the channel count from all the guilds the bot is connected to.

async static get_cooldown_time(*time*)
Turn command cooldown of seconds into hours, minutes, and seconds.

async get_disabled_server_interactions(*server_id*)
Get a server's disabled interactions.

static get_int_index(*number*, *index*)
Retrieves the specific index of an integer. Ex: Calling index 3 for integer 12345 will return 123.

async get_language_code(*input_language*)
Returns a language code that is compatible with the papago framework.

get_server_count()
Returns the guild count the bot is connected to.

get_text_channel_count()
Returns the text channel count from all the guilds the bot is connected to.

get_user_count()
Get the amount of users that the bot is watching over.

get_voice_channel_count()
Returns the voice channel count from all the guilds the bot is connected to.

async send_ban_message(*channel*)
A message to send for a user that is banned from the bot.

async unban_user_from_bot(*user_id*)
UnBans a user from the bot.

4.15 u_moderator

class IreneUtility.util.u_moderator.**Moderator**(**args*)

async add_welcome_message_server(*channel_id*, *guild_id*, *message*, *enabled*)
Adds a new welcome message server.

async check_welcome_message_enabled(*server_id*)
Check if a welcome message server is enabled.

async toggle_games(*channel_id: int*) → bool
Toggles game usage in a text channel.

Will return True if channel has games enabled.

async update_welcome_message_channel(*server_id*, *channel_id*)
Update the welcome message channel.

async update_welcome_message_enabled(*server_id*, *enabled*)
Update a welcome message server's enabled status

4.16 u_patreon

class IreneUtility.util.u_patreon.**Patreon**(**args*)

async add_to_patreon(*user_id*)
Add user as a permanent patron.

async check_if_patreon(*user_id*, *super_patron=False*)
Check if the user is a patreon. There are two ways to check if a user ia a patreon. The first way is getting the members in the Patreon/Super Patreon Role. The second way is a table to check for permanent patreon users that are directly added by the bot owner. – After modifying -> We take it straight from cache now.

async get_patreon_role_members(*super_patron=False*)
Get the members in the patreon roles.

async get_patreon_users()
Get the permanent patron users

async remove_from_patreon(*user_id*)
Remove user from being a permanent patron.

async reset_patreon_cooldown(*ctx*)
Checks if the user is a patreon and resets their cooldown.

4.17 u_reminder

class IreneUtility.util.u_reminder.**Reminder**(**args*)

async static determine_time_type(*user_input*)
Determine if time is relative time or absolute time relative time: remind me to _____ in 6 days absolute time: remind me to _____ at 6PM

async static format_time(*string_format*, *user_timezone*, *input_time: Optional[datetime.datetime] = None*)
Format time according to the user timezone

async get_locale_time(*m_time*, *user_timezone=None*)
Return a string containing locale date format. For now, enforce all weekdays to be en_US format

async get_reminders(*user_id*)
Get the reminders of a user

async get_user_timezone(*user_id*)
Returns the user's timezone

async process_absolute_time_input(*time_input*, *user_id*)
Returns the absolute date time of the input

async process_relative_time_input(*time_input*)
Returns the relative time of the input in seconds

async static process_reminder_reason(*user_input*, *cutoff_index*)
Return the reminder reason that comes before in/at

async process_reminder_time(*user_input*, *type_index*, *is_relative_time*, *user_id*)
Return the datetime of the reminder depending on the time format

async static process_timezone_input(*input_timezone*, *input_country_code=None*)
Convert timezone abbreviation and country code to standard timezone name

async remove_user_reminder(*user_id*, *reminder_id*)
Remove a reminder from the cache and the database.

async remove_user_timezone(*user_id*)
Remove user timezone

async set_reminder(*remind_reason*, *remind_time*, *user_id*)
Add reminder date to cache and db.

async set_user_timezone(*user_id*, *timezone*)
Set user timezone

4.18 u_selfassignroles

class IreneUtility.util.u_selfassignroles.**SelfAssignRoles**(**args*)

async add_self_role(*role_id*, *role_name*, *server_id*)
Adds a self-assignable role to a server.

async check_for_self_assignable_role(*message*)
Main process for processing self-assignable roles.

async static check_member_has_role(*member_roles*, *role_id*)
Check if a member has a role

async check_self_assignable_channel(*server_id*, *channel*)
Check if a channel is a self assignable role channel.

async check_self_role_exists(*role_id*, *role_name*, *server_id*)
Check if a role exists as a self-assignable role in a server.

async get_assignable_server_roles(*server_id*)
Get all the self-assignable roles from a server.

async get_self_role(*message_content*, *server_id*)
Returns a discord.Object that can be used for adding or removing a role to a member.

async modify_channel_role(*channel_id*, *server_id*)
Add or Change a server's self-assignable role channel.

async process_member_roles(*message*, *role*, *role_name*, *prefix*, *author*)
Adds or removes a (Self-Assignable) role from a member

async remove_current_channel_role(*channel_id*, *server_id*)
Remove the self-assignable role channel if the current channel was previously assigned.

async remove_self_role(*role_name*, *server_id*)
Remove a self-assignable role from a server.

4.19 u_twitch

class IreneUtility.util.u_twitch.**Twitch**(**args*)

async add_channel(*twitch_username*, *guild_id*)
Follows a twitch channel.

async change_twitch_role(*guild_id*, *role_id*)
Adds/Changes a twitch role that gets mentioned on twitch updates.

async check_channel_followed(*twitch_username*, *guild_id*)
Check if a guild is being followed.

async check_guild_limit(*guild_id*)
check if a guild is allowed to follow more channels

async delete_twitch_role(*guild_id*)
Delete a twitch role mentioned on updates.

async get_channels_followed(*guild_id*)
Get the twitch channels a discord server follows.

async get_discord_channel(*guild_id*)
Get the channel that follow announcements are sent to.

async remove_channel(*twitch_username*, *guild_id*)
Stop following a twitch channel.

async reset_twitch_token()
Get/and reset twitch access token to use on the twitch api.

async set_discord_channel(*guild_id*, *channel_id*)
Set the channel for a guild that receives live updates.

4.20 u_twitter

class IreneUtility.util.u_twitter.**Twitter**(**args*)

get_random_idol_photo()
Get a random idol photo existing in the file directory.

This method may block the heartbeat due to OS operation. Should be run separately.

async upload_random_image()
Uploads a random (BUT UNIQUE) idol photo to twitter.

Returns twitter body message & twitter link to the post.

4.21 u_weverse

class IreneUtility.util.u_weverse.**Weverse**(**args*)

async add_weverse_channel(*channel_id*, *community_name*)
Add a channel to get updates for a community

async add_weverse_channel_to_cache(*channel_id*, *community_name*)
Add a weverse channel to cache.

async add_weverse_role(*channel_id*, *community_name*, *role_id*)
Add a weverse role to notify.

async change_weverse_comment_media_status(*channel_id*, *community_name*, *t_disabled*, *updated=False*, *media=False*)
Change a channel's subscription and whether or not they receive updates on comments/comments.

Parameters

- **channel_id** – (int) Channel id on discord
- **community_name** – (str) Community name on Weverse
- **t_disabled** – (integer) Represents the current status of the disable.
- **updated** – (bool) Whether it needs to be updated.
- **media** – (bool) Whether to disable media or not.

async change_weverse_media_status(*channel_id*, *community_name*, *media_disabled*, *updated=False*)
Change a channel's subscription and whether or not they receive updates on media.

async check_weverse_channel(*channel_id*, *community_name*)
Check if a channel is already getting updates for a community

async delete_weverse_channel(*channel_id*, *community_name*)
Delete a community from a channel's updates.

async delete_weverse_role(*channel_id*, *community_name*)
Remove a weverse role from a server (no longer notifies a role).

async disable_type(*ctx*, *community_name*, *media=False*)
Disable media/comments on a community and deal with the user messages.

Parameters

- **ctx** – Context Object
- **community_name** – Weverse community name
- **media** – Whether the post type is media

async download_weverse_post(*url*, *file_name*)
Downloads an image url and returns image host url.

If we are to upload from host, it will return the folder location instead.

async get_weverse_channels(*community_name*)
Get all of the channel ids for a specific community name

async replace_cache_role_id(*channel_id*, *community_name*, *role_id*)
Replace the server role that gets notified on Weverse Updates.

async set_comment_embed(*notification*, *embed_title*)
Set Comment Embed for Weverse.

async set_media_embed(*notification*, *embed_title*)
Set Media Embed for Weverse.

async set_post_embed(*notification*, *embed_title*)
Set Post Embed for Weverse.

Returns Embed and (a list of file locations OR a string with image urls)

CHAPTER

FIVE

S_SQL

5.1 __init__ (s_sql)

class IreneUtility.s_sql.**SqlConnection**

Used so that we have a stable reference to our DB Connection. This way we do not need to worry if our connection at any starting point is None as long as it gets set eventually

self - The primary instance of *SqlConnection*

5.2 db_structure

async IreneUtility.s_sql.db_structure.**create_db_structure**()

Creates the db structure based on the existing version sql file.

This prevents having the manually update the db structure across development bots. Before this method is used, there must be a SQL connection whether it is connected to another DB, or creating an empty DB with no schemas just to create the python connection.

SQL File must be in the main directory of the Bot Client.

Includes a blocking File IO Task, but since this is executed on start/run, it really does not matter.

5.3 s_biasgame

5.4 s_blackjack

async IreneUtility.s_sql.s_blackjack.**delete_playing_cards**()

Delete all custom playing cards from table.

async IreneUtility.s_sql.s_blackjack.**fetch_playing_cards**()

Fetch playing cards.

async IreneUtility.s_sql.s_blackjack.**generate_playing_card**(*card_value_id*, *bg_idol_id*) → int

Add a playing card and return the custom id.

Note that we could technically have a file format name of "(card id)_(idol id).png", but in the case we would like to add several images for one idol, it would be better that we go by the unique index of the table, even if it does take two more sql queries than needed be.

Parameters

- **card_value_id** – Number from 1 to 52 that represents the custom card.
- **bg_idol_id** – Idol ID of the background.

Returns Custom ID of card.

5.5 s_cache

async IreneUtility.s_sql.s_cache.**add_guild**(*guild*)
Adds a guild to the guild table.

Parameters **guild** – D.py Guild

async IreneUtility.s_sql.s_cache.**remove_guild**(*guild*)
Removes a guild from the guild table.

Parameters **guild** – D.py Guild

5.6 s_general

async IreneUtility.s_sql.s_general.**delete_welcome_role**(*guild_id: int*)
Delete a guild's welcome role.

async IreneUtility.s_sql.s_general.**fetch_bot_bans**()
Fetch all bot bans.

async IreneUtility.s_sql.s_general.**fetch_bot_statuses**()
Fetch all bot statuses.

async IreneUtility.s_sql.s_general.**fetch_mod_mail**()
Fetch mod mail users and channels.

async IreneUtility.s_sql.s_general.**fetch_n_word**(*ordered_by_greatest=False*)
Fetch all users N Word count.

async IreneUtility.s_sql.s_general.**fetch_server_prefixes**()
Fetch the server prefixes.

async IreneUtility.s_sql.s_general.**fetch_temp_channels**()
Fetch all temporary channels

async IreneUtility.s_sql.s_general.**fetch_welcome_messages**()
Fetch all welcome messages

async IreneUtility.s_sql.s_general.**fetch_welcome_roles**()
Fetch all welcome roles.

async IreneUtility.s_sql.s_general.**insert_welcome_role**(*guild_id: int*, *role_id: int*)
Insert or Update a welcome role.

async IreneUtility.s_sql.s_general.**update_welcome_role**(*guild_id: int*, *role_id: int*)
Update a guild's welcome role.

5.7 s_groupmembers

async IreneUtility.s_sql.s_groupmembers.**delete_send_idol_photo_channel**(*text_channel_id: int*)
Deletes a text channel from receiving photos after t time.

Parameters **text_channel_id** – ID of the text channel that should no longer receive idol photos.

async IreneUtility.s_sql.s_groupmembers.**fetch_aliases**(*object_id*, *group=False*)
Fetch all global and server aliases of an idol or group.

Parameters

- **object_id** – An Idol or Group id
- **group** – Whether the object is a group.

async IreneUtility.s_sql.s_groupmembers.**fetch_all_groups**()
Fetch all groups.

async IreneUtility.s_sql.s_groupmembers.**fetch_all_idols**()
Fetch all idols.

async IreneUtility.s_sql.s_groupmembers.**fetch_all_images**()
Fetch all images.

async IreneUtility.s_sql.s_groupmembers.**fetch_dead_links**()
Fetch all dead links.

async IreneUtility.s_sql.s_groupmembers.**fetch_members_in_group**(*group_id*)
Fetches the idol ids in a group.

Parameters **group_id** – The group's id

async IreneUtility.s_sql.s_groupmembers.**fetch_restricted_channels**()
Fetch all restricted idol photo channels.

async IreneUtility.s_sql.s_groupmembers.**fetch_send_idol_photos**()
Fetches the text channels and idols that should be sent to the channel after t time.

async IreneUtility.s_sql.s_groupmembers.**get_idol_id_by_image_id**(*image_id: int*)
Get an idol id from a unique image id.

Returns Idol ID or NoneType (if the image id does not exist)

async IreneUtility.s_sql.s_groupmembers.**insert_send_idol_photo**(*text_channel_id: int*, *idol_id: int*)
Inserts a text channel to receive photos from certain idols.

Parameters

- **text_channel_id** – ID of the text channel that will receive idol photos from the idol.
- **idol_id** – The idol id that will have their photos be sent to the text channel.

async IreneUtility.s_sql.s_groupmembers.**update_send_idol_photo**(*text_channel_id: int*, *idol_ids: list*)
Update a text channel's idol list

Parameters

- **text_channel_id** – ID of the text channel that will receive idol photos from the idol.
- **idol_ids** – ALL idol ids that should be associated with the text channel

5.8 s_guessinggame

async IreneUtility.s_sql.s_guessinggame.**fetch_filter_enabled**()
Fetches the users with a guessing game filter enabled.

async IreneUtility.s_sql.s_guessinggame.**fetch_filtered_groups**()
Fetches all the users and the groups they have filtered.

async IreneUtility.s_sql.s_guessinggame.**fetch_gg_stats**()
Fetch the user's id, easy, medium, and hard guessing game stats

5.9 s_lastfm

5.10 s_levels

async IreneUtility.s_sql.s_levels.**create_level_row**(*user_id: int*)
Create a row in currency.levels for the user.

Parameters **user_id** – Discord User ID

async IreneUtility.s_sql.s_levels.**fetch_levels**()
Fetches all user ids and their rob, daily, beg, and profile level.

async IreneUtility.s_sql.s_levels.**get_profile_xp**(*user_id: int*)
Get a user's profile xp.

Parameters **user_id** – Discord User ID

async IreneUtility.s_sql.s_levels.**level_row_exists**(*user_id: int*)
Check if a user has a row in the levels table.

async IreneUtility.s_sql.s_levels.**update_level**(*user_id: int*, *column_name: str*, *level: int*)
Update the level of a user.

Parameters

- **user_id** – Discord User ID
- **column_name** – Column name of the currency.levels table
- **level** – Level to set column to.

5.11 s_logging

async IreneUtility.s_sql.s_logging.**fetch_logged_channels**(*primay_key*)
Fetch the channels of a logged server.

Parameters **primay_key** – The table key of the logged server.

async IreneUtility.s_sql.s_logging.**fetch_logged_servers**()
Fetch the servers being logged.

5.12 s_miscellaneous

5.13 s_moderator

async IreneUtility.s_sql.s_moderator.**disable_game_in_channel**(*channel_id: int*)
Disable games in a text channel.

async IreneUtility.s_sql.s_moderator.**enable_game_in_channel**(*channel_id: int*)
Enable games in a text channel.

async IreneUtility.s_sql.s_moderator.**fetch_games_disabled**()
Fetch the servers being logged.

5.14 s_patreon

async IreneUtility.s_sql.s_patreon.**add_patron**(*user_id*, *super_patron: int*)
Add a patron

Parameters

- **user_id** – Discord User ID
- **super_patron** – 1 for the user becoming a super patron, or 0 if they are a normal patron.

async IreneUtility.s_sql.s_patreon.**delete_patron**(*user_id*)
Delete a patron.

Parameters **user_id** – Discord User ID

async IreneUtility.s_sql.s_patreon.**fetch_cached_patrons**()
Fetch the cached patrons.

async IreneUtility.s_sql.s_patreon.**update_patron**(*user_id*, *super_patron: int*)
Updates a patron's status

Parameters

- **user_id** – Discord User ID
- **super_patron** – 1 for the user becoming a super patron, or 0 if they are a normal patron.

5.15 s_reminder

async IreneUtility.s_sql.s_reminder.**fetch_reminders**()
Fetch all reminders. (id, user id, reason, timestamp)

5.16 s_selfassignroles

async IreneUtility.s_sql.s_selfassignroles.**fetch_all_self_assign_channels**()
Fetch all channel ids and server ids for self assignable roles.

async IreneUtility.s_sql.s_selfassignroles.**fetch_all_self_assign_roles**()
Fetch all role ids, role names, and server ids for self assignable roles.

5.17 s_session

async IreneUtility.s_sql.s_session.**add_new_session**(*total_used*, *session_commands*, *date*)
Insert a new session.

Parameters

- **total_used** – Total commands used for all sessions.
- **session_commands** – The amount of commands to give the session (usually 0)
- **date** – Usually datetime.date.today()

async IreneUtility.s_sql.s_session.**fetch_command**(*session_id*)
Fetch the command name and its usage amount for a certain session.

async IreneUtility.s_sql.s_session.**fetch_session_id**(*date*)
Fetch a session id with a date.

Parameters **date** – Usually datetime.date.today()

async IreneUtility.s_sql.s_session.**fetch_session_usage**(*date*)
Fetch the session command usage of a date.

async IreneUtility.s_sql.s_session.**fetch_total_session_usage**()
Fetches the total amount of commands used.

5.18 s_twitch

async IreneUtility.s_sql.s_twitch.**check_twitch_already_posted**(*twitch_username*, *channel_id*) → bool
Check if a twitch channel being live was already posted to a text cahnnel.

Parameters

- **twitch_username** – Twitch username
- **channel_id** – Text Channel ID

Returns True if the live announcement was already posted.

async IreneUtility.s_sql.s_twitch.**delete_twitch_posted**(*twitch_username*)
Delete the text channels from a table that have received messages for a twitch username. :param twitch_username: Twitch username

async IreneUtility.s_sql.s_twitch.**fetch_twitch_guilds**()
Fetch all guild ids, channel ids, and role ids

async IreneUtility.s_sql.s_twitch.**fetch_twitch_notifications**()
Fetch all twitch username and guild ids that announcements should be sent to.

async IreneUtility.s_sql.s_twitch.**set_twitch_posted**(*twitch_username*, *channel_id*)
Set a discord text channel to have already been sent a message from twitch announcements. :param twitch_username: Twitch username :param channel_id: Text Channel ID

5.19 s_twitter

async IreneUtility.s_sql.s_twitter.**check_photo_uploaded**(*image_id*)
Checks if a photo was already uploaded.

Parameters **image_id** – The unique image id.

Returns Count of the image id in the table (should be 0 or 1)

async IreneUtility.s_sql.s_twitter.**insert_photo_uploaded**(*image_id*, *media_id*)
Insert an image so that we can keep track of twitter media uploads.

Parameters

- **image_id** – Unique Image ID
- **media_id** – Twitter Media ID

5.20 s_user

async IreneUtility.s_sql.s_user.**delete_user_language**(*user_id: int*)
Deletes the user from the language table. Default is en_us [Which should not exist in the language table]

Parameters **user_id** –

async IreneUtility.s_sql.s_user.**fetch_languages**()
Fetches all user ids and their language preference.

async IreneUtility.s_sql.s_user.**fetch_timezones**()
Fetch all timezones. (user id, timezone)

async IreneUtility.s_sql.s_user.**set_user_language**(*user_id: int*, *language: str*)
Set the user's client language.

Parameters

- **user_id** –
- **language** –

5.21 s_weverse

async IreneUtility.s_sql.s_weverse.**fetch_weverse**()
Fetch all weverse subscriptions.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W

X