Weverse

MujyKun

May 20, 2021

CONTENTS:

WEVERSECLIENT

class `Weverse.`**`WeverseClient`**(***kwargs*)

    Abstract & Parent Client for connecting to Weverse and creating the internal cache.

    Do not create an object directly from this class. Instead, create a *`Weverse.WeverseClientSync`* or *`Weverse.WeverseClientAsync`* object since those are concrete.

        Parameters

- **verbose** (*bool*) – Whether to print out verbose messages.

- **web_session** – An aiohttp or requests client session.

- **token** – The account token to connect to the Weverse API. In order to find your token, please refer to *Get Account Token*

    verbose

        Whether to print out verbose messages.

            Type bool

    web_session

        An aiohttp or requests client session.

    token

        The account token to connect to the Weverse API. In order to find your token, please refer to *Get Account Token*

    user_notifications

        Most recent notifications of the account connected.

            Type list

    headers

        API Header

            Type dict

    api_url

        URL to connect to the API

            Type str

    api_communities_url

        Endpoint for communities

            Type str

    api_notifications_url

        Endpoint for user notifications

> **Type** str

api_new_notifications_url
> Endpoint for checking new user notifications

> > **Type** str

api_all_artist_posts_url
> Endpoint for getting the Artist Feed from a community

> > **Type** str

api_artist_to_fans
> part of the endpoint (NOT FULL) for viewing the tofans posts.

> > **Type** str

api_all_communities_info_url
> Endpoint for information about ALL communities and ALL idols.

> > **Type** str

cache_loaded
> Whether the Internal Weverse Cache is fully loaded. This will change for a split moment when grabbing a new post.

> > **Type** bool

new_media
> We do not store ALL old media objects as cache, so only when there are new media, we store it

> > **Type** list(*Media*)

user_endpoint
> User info endpoint.

> > **Type** str

all_posts
> All posts in cache where the dict of Post ID is the key, and the value being the Post Object

> > **Type** dict(*Post*)

all_artists
> All artists in cache where the dict of Artist ID is the key, and the value being the Artist Object

> > **Type** dict(*Artist*)

all_comments
> All comments in cache where the dict of Comment ID is the key, and the value being the Comment Object

> > **Type** dict(*Comment*)

all_notifications
> All notifications in cache where the dict of Notification ID is the key, and the value being the Notification Object

> > **Type** dict(*Notification*)

all_photos
> All photos in cache where the dict of Photo ID is the key, and the value being the Photo Object

> > **Type** dict(*Photo*)

all_communities

All communities in cache where the dict of Community ID is the key, and the value being the Community Object

> **Type** dict(*Community*)

all_media

All media in cache where the dict of Media ID is the key, and the value being the Media Object

> **Type** dict(*Media*)

all_tabs

All tabs in cache where the dict of Tab ID is the key, and the value being the Tab Object

> **Type** dict(*Tab*)

check_status(*status*, *url*) → bool

Confirm the status of a URL

> **Parameters**
>
> - **status** – Status code of url connection
>
> - **url** – Link that we connected to.
>
> **Returns** True if the connection was a success.
>
> **Raises** *Invalid Token* if there was an invalid token.

static determine_notification_type(*notification_body*) → str

Determine the post type based on the notification body.

Since notifications do not differentiate between Posts and Comments, this is for that purpose.

> **Parameters** **notification_body** – The body of the notification.
>
> **Returns** A string with either "comment", "media", or "post"

get_artist_by_id(*artist_id*) → *Weverse.models.artist.Artist*

Get artist by their ID.

> **Parameters** **artist_id** – The artist's ID
>
> **Returns** *Artist* or NoneType

get_comment_by_id(*comment_id*) → *Weverse.models.comment.Comment*

Get a comment by the ID :param comment_id: Comment ID :returns: *Comment* or NoneType

get_community_by_id(*community_id*) → *Weverse.models.community.Community*

Get a community by the ID.

> **Parameters** **community_id** – Community ID
>
> **Returns** *Community* or NoneType

get_media_by_id(*media_id*) → *Weverse.models.media.Media*

Get Media by the ID

> **Parameters** **media_id** – Media ID
>
> **Returns** *Media* or NoneType

get_notification_by_id(*notification_id*) → *Weverse.models.notification.Notification*

Get a notification by the ID

> **Parameters** **notification_id** – Notification ID
>
> **Returns** *Notification* or NoneType

get_photo_by_id(*photo_id*) → *Weverse.models.photo.Photo*
    Get a photo by the ID

        Parameters `photo_id` – Photo ID

        Returns *Photo* or NoneType

get_post_by_id(*post_id*) → *Weverse.models.post.Post*
    Get a post by the ID

        Parameters `post_id` – Post ID

        Returns *Post* or NoneType

get_tab_by_id(*tab_id*) → *Weverse.models.tab.Tab*
    Get tab by their ID.

        Parameters `tab_id` – The tab ID

        Returns *Tab* or NoneType

static `process_community_artists_and_tabs`(*community*, *response_text_as_dict*)
    Process the community artists and tabs and add them to their respective communities.

        Parameters

            • `community` – Community object

            • `response_text_as_dict` – Response text of connection to endpoint but as a dict.

CLIENTS

2.1 WeverseClientSync

class Weverse.**WeverseClientSync**(***kwargs*)

Synchronous Weverse Client that Inherits from *WeverseClient*.

> **Parameters kwargs** – Same as *WeverseClient*.

Attributes are the same as *WeverseClient*.

check_new_user_notifications()

Checks if there is a new user notification, updates the cache, and returns if there was.

> **Returns** (bool) Whether there is a new notification.

check_token_works()

Check if a token is invalid.

> **Returns** (bool) True if the token works.

create_communities()

Get and Create the communities the logged in user has access to.

create_community_artists_and_tabs()

Create the community artists and tabs and add them to their respective communities.

create_post(*community:* Weverse.models.community.Community, *post_id*) → *Weverse.models.post.Post*

Create a post and update the cache with it. This is meant for an individual post.

> **Parameters**
>
> - **community** – *Community* the post was created under.
>
> - **post_id** – The id of the post we are needing to fetch.

create_posts(*community:* Weverse.models.community.Community, *next_page_id: Optional[int] = None*)

Paginate through a community's posts and add it to object cache.

> **Parameters**
>
> - **community** – *Community* the posts exist under.
>
> - **next_page_id** (*[OPTIONAL]*) – Next Page ID (Weverse paginates posts).

fetch_artist_comments(*community_id*, *post_id*)

Fetches the artist comments on a post.

> **Parameters**
>
> - **community_id** – Community ID the post is on.

- **post_id** – Post ID to fetch the artist comments of.

> **Returns** List[*Comment*]

fetch_comment_body(*community_id*, *comment_id*)
    Fetches a comment from its ID.

> **Parameters**
>
> - **community_id** – The ID of the community the comment belongs to.
>
> - **comment_id** – The ID of the comment to fetch.
>
> **Returns** (str) Body of the comment.

fetch_media(*community_id*, *media_id*)
    Receive media object based on media id.

> **Parameters**
>
> - **community_id** – The ID of the community the media belongs to.
>
> - **media_id** – The ID of the media to fetch.
>
> **Returns** *Media* or NoneType

get_user_notifications()
    Get a list of updated user notification objects.

> **Returns** List[*Notification*]

start()
    Creates internal cache.

    This is the main process that should be run.

> **Raises** `Weverse.error.PageNotFound`
>
> **Raises** `Weverse.error.InvalidToken`
>
> **Raises** `Weverse.error.BeingRateLimited`

translate(*post_or_comment_id*, *is_post=False*, *is_comment=False*, *p_obj=None*, *community_id=None*)
    Translates a post or comment, must set post or comment to True.

> **Parameters**
>
> - **post_or_comment_id** – A post or comment ID.
>
> - **is_post** (`[OPTIONAL]`) – If we passed in a post.
>
> - **is_comment** (`[OPTIONAL]`) – If we passed in a comment
>
> - **p_obj** (`[OPTIONAL]`) – The object we are looking to translate
>
> - **community_id** (`[OPTIONAL]`) – The community id the post/comment was made under.
>
> **Returns** (str) Translated message or NoneType

update_cache_from_notification()
    Grab a new post based from a notification and add it to cache.

2.2 WeverseClientAsync

class Weverse.**WeverseClientAsync**(*loop=<_UnixSelectorEventLoop running=False closed=False debug=False>, **kwargs*)

Asynchronous Weverse Client that Inherits from *WeverseClient*.

> **Parameters**
>
> - **loop** – Asyncio Event Loop
> - **kwargs** – Same as Weverse.

loop

Asyncio Event Loop

Attributes are the same as :ref:`WeverseClient`.

async **check_new_user_notifications**() → bool

Checks if there is a new user notification, updates the cache, and returns if there was.

This is a coroutine and must be awaited.

> **Returns** (bool) Whether there is a new notification.

async **check_token_works**() → bool

Check if a token is invalid.

This is a coroutine and must be awaited.

> **Returns** (bool) True if the token works.

async **create_communities**()

Get and Create the communities the logged in user has access to.

This is a coroutine and must be awaited.

async **create_community_artists_and_tabs**()

Create the community artists and tabs and add them to their respective communities.

This is a coroutine and must be awaited.

async **create_post**(*community:* Weverse.models.community.Community, *post_id*) → *Weverse.models.post.Post*

Create a post and update the cache with it. This is meant for an individual post.

This is a coroutine and must be awaited.

> **Parameters**
>
> - **community** – *Community* the post was created under.
> - **post_id** – The id of the post we are needing to fetch.

async **create_posts**(*community:* Weverse.models.community.Community, *next_page_id: Optional[int] = None*)

Paginate through a community's posts and add it to object cache.

This is a coroutine and must be awaited.

> **Parameters**
>
> - **community** – *Community* the posts exist under.
> - **next_page_id** (*[OPTIONAL]*) – Next Page ID (Weverse paginates posts).

async **fetch_artist_comments**(*community_id*, *post_id*)
> Fetches the artist comments on a post.
>
> This is a coroutine and must be awaited.
>
> > **Parameters**
> >
> > - **community_id** – Community ID the post is on.
> >
> > - **post_id** – Post ID to fetch the artist comments of.
> >
> > **Returns** List[*Comment*]

async **fetch_comment_body**(*community_id*, *comment_id*)
> Fetches a comment from its ID.
>
> This is a coroutine and must be awaited.
>
> > **Parameters**
> >
> > - **community_id** – The ID of the community the comment belongs to.
> >
> > - **comment_id** – The ID of the comment to fetch.
> >
> > **Returns** (str) Body of the comment.

async **fetch_media**(*community_id*, *media_id*)
> Receive media object based on media id.
>
> This is a coroutine and must be awaited.
>
> > **Parameters**
> >
> > - **community_id** – The ID of the community the media belongs to.
> >
> > - **media_id** – The ID of the media to fetch.
> >
> > **Returns** *Media* or NoneType

async **get_user_notifications**()
> Get a list of updated user notification objects.
>
> This is a coroutine and must be awaited.
>
> > **Returns** List[*Notification*]

async **start**()
> Creates internal cache.
>
> This is the main process that should be run.
>
> This is a coroutine and must be awaited.
>
> > **Raises** *Weverse.error.PageNotFound*
> >
> > **Raises** *Weverse.error.InvalidToken*
> >
> > **Raises** *Weverse.error.BeingRateLimited*

async **translate**(*post_or_comment_id*, *is_post=False*, *is_comment=False*, *p_obj=None*, *community_id=None*)
> Translates a post or comment, must set post or comment to True.
>
> This is a coroutine and must be awaited.
>
> > **Parameters**
> >
> > - **post_or_comment_id** – A post or comment ID.

- **is_post** (*[OPTIONAL]*) – If we passed in a post.

- **is_comment** (*[OPTIONAL]*) – If we passed in a comment

- **p_obj** (*[OPTIONAL]*) – The object we are looking to translate

- **community_id** (*[OPTIONAL]*) – The community id the post/comment was made under.

   Returns (str) Translated message or NoneType

async update_cache_from_notification()
   Grab a new post based from a notification and add it to cache.

   This is a coroutine and must be awaited.

MODELS

3.1 Community

class Weverse.models.**Community**(***kwargs*)

> A Comment object that represents a Weverse Comment that belongs to an Artist.
>
> It is not suggested to create a Comment manually, but rather through the following method: *Weverse.objects.create_comment_objects*
>
> The information retrieved on a Comment is directly from the Weverse API and altered to fit this class.
>
> > **Parameters**
> >
> > - **id** (int) – The Community ID.
> > - **name** (str) – The Community Name.
> > - **description** (str) – Description of the Community.
> > - **member_count** (int) – Amount of members in the community.
> > - **home_banner** (str) – Direct Image URL to the home banner.
> > - **icon** (str) – Direct Image URL to the Icon.
> > - **Banner** (str) – Direct Image URL to the Banner.
> > - **full_name** (str) – Full Name of the Community.
> > - **fc_member** (*bool*) – If a special membership is required to join.
> > - **show_member_count** (*bool*) – If the member count is visible.
>
> **id**
> > The Community ID.
> >
> > > **Type** int
>
> **name**
> > The Community Name.
> >
> > > **Type** str
>
> **description**
> > Description of the Community.
> >
> > > **Type** str
>
> **member_count**
> > Amount of members in the community.

Type int

home_banner
>   Direct Image URL to the home banner.

>   **Type** str

icon
>   Direct Image URL to the Icon.

>   **Type** str

Banner
>   Direct Image URL to the Banner.

>   **Type** str

full_name
>   Full Name of the Community.

>   **Type** str

fc_member
>   If a special membership is required to join.

>   **Type** bool

show_member_count
>   If the member count is visible.

>   **Type** bool

artists
>   List of artists the community has.

>   **Type** List[*Artist*]

tabs
>   The Tabs the community has.

>   **Type** List[*Tab*]

3.2 Notification

class Weverse.models.**Notification**(**kwargs*)
>   A Media object that represents a Weverse Media Post.

>   It is not suggested to create a Notification object manually, but rather through the following method: *Weverse.objects.create_notification_objects*

>   The information retrieved on a Notification is directly from the Weverse API and altered to fit this class.

>   **Parameters**

>   - **id** (*int*) – The id of the notification.

>   - **message** (*str*) – The message of the notification.

>   - **bold_element** (*str*) – The bolded element in the notification.

>   - **community_id** (*int*) – The community id associated with the notification.

>   - **community_name** (*str*) – The community name associated with the notification.

>   - **contents_type** (*str*) – The type of post it is.

- **contents_id** (*int*) – The id of the content post.

- **notified_at** – The time the notification was triggered.

- **icon_image_url** (*str*) – Icon image url of the notification.

- **thumbnail_image_url** (*str*) – Thumbnail url of the notification.

- **artist_id** (*int*) – The ID of the Artist that released the content.

- **is_membership_content** (*bool*) – If the content is exclusive to members.

- **is_web_only** (*bool*) – Whether the notification is only available directly on the website.

- **platform** (*str*) – The platform of the notification.

id
> The id of the notification.
>
> > **Type** int

message
> The message of the notification.
>
> > **Type** str

bold_element
> The bolded element in the notification.
>
> > **Type** str

community_id
> The community id associated with the notification.
>
> > **Type** int

community_name
> The community name associated with the notification.
>
> > **Type** str

contents_type
> The type of post it is.
>
> > **Type** str

contents_id
> The id of the content post.
>
> > **Type** int

notified_at
> The time the notification was triggered.

icon_image_url
> Icon image url of the notification.
>
> > **Type** str

thumbnail_image_url
> Thumbnail url of the notification.
>
> > **Type** str

artist_id
> The ID of the Artist that released the content.
>
> > **Type** int

`is_membership_content`
>   If the content is exclusive to members.
>
>>   **Type** bool

`is_web_only`
>   Whether the notification is only available directly on the website.
>
>>   **Type** bool

`platform`
>   The platform of the notification.
>
>>   **Type** str

3.3 Photo

class Weverse.models.**Photo**(***kwargs*)
>   A Photo object that represents a Weverse Photo that belongs to media or a post.
>
>   It is not suggested to create a Photo manually, but rather through the following method: *Weverse.objects.*
>   *create_photo_objects*
>
>   The information retrieved on a Photo is directly from the Weverse API and altered to fit this class.
>
>   **Parameters**
>
>   - **id** (*int*) – The ID of the photo.
>
>   - **content_index** (*int*) – Index the photo is in from a bundle of photos.
>
>   - **thumbnail_img_url** (*str*) – The thumbnail image link.
>
>   - **thumbnail_img_width** (*str*) – The original image width.
>
>   - **thumbnail_img_height** (*str*) – The thumbnail image height.
>
>   - **original_img_url** (*str*) – The original image link.
>
>   - **original_img_width** (*str*) – The original image width.
>
>   - **original_img_height** (*str*) – The original image height.
>
>   - **file_name** (*str*) – File name of the photo.
>
>   `id`
>   >   The ID of the photo.
>   >
>   >>   **Type** int
>
>   `content_index`
>   >   Index the photo is in from a bundle of photos.
>   >
>   >>   **Type** int
>
>   `thumbnail_img_url`
>   >   The thumbnail image link.
>   >
>   >>   **Type** str
>
>   `thumbnail_img_width`
>   >   The original image width.
>   >
>   >>   **Type** str

thumbnail_img_height
>   The thumbnail image height.

>   **Type** str

original_img_url
>   The original image link.

>   **Type** str

original_img_width
>   The original image width.

>   **Type** str

original_img_height
>   The original image height.

>   **Type** str

file_name
>   File name of the photo.

>   **Type** str

post
>   The Post Object the photo belongs to.

>   **Type** *Post*

3.4 Artist

class Weverse.models.**Artist**(***kwargs*)
>   An Artist object that represents a Weverse Artist that belongs in a community.

>   It is not suggested to create an Artist manually, but rather through the following method: *Weverse.objects.*
>   *create_artist_objects*

>   The information retrieved on an Artist is directly from the Weverse API and altered to fit this class.

>   **Parameters**

>> • **id** (int) – The Artist ID.

>> • **community_user_id** (int) – Artist's ID in the community.

>> • **name** (str) – The Primary Artist Name.

>> • **list_name** (list) – A list of names for the Artist.

>> • **is_online** (bool) – Whether the Artist is currently online

>> • **profile_nick_name** (str) – Artist nickname.

>> • **profile_img_path** (str) – Image URL for the Artist's profile.

>> • **is_birthday** (bool) – Whether it is the Artist's birthday.

>> • **group_name** (str) – The group name the Artist is associated with.

>> • **max_comment_count** (int) – The maximum amount of comments this Artist can post.

>> • **community_id** (int) – The ID of the community this Artist object was selected from.

>> • **is_enabled** (bool) – If the Artist account is enabled.

- **has_new_to_fans** (`bool`) – If the Artist has a new post for fans.

- **has_new_private_to_fans** (`bool`) – If the Artist has a new private post for fans.

- **to_fan_last_id** (`int`) – The latest tofan post ID.

- **to_fan_last_created_at** – When the artist's last tofan post was created.

- **to_fan_last_expire_in** – When the artist's last tofan post expires.

- **birthday_img_url** (`str`) – A direct image url to the artist's birthday image.

- **community** (`Community`) – The community the Artist is in.

- **posts** (`list`) – A list of posts the Artist has.

id

The Artist ID.

> **Type** int

community_user_id

Artist's ID in the community.

> **Type** int

name

The Primary Artist Name.

> **Type** str

list_name

A list of names for the Artist.

> **Type** list

is_online

Whether the Artist is currently online

> **Type** bool

profile_nick_name

Artist nickname.

> **Type** str

profile_img_path

Image URL for the Artist's profile.

> **Type** str

is_birthday

Whether it is the Artist's birthday.

> **Type** bool

group_name

The group name the Artist is associated with.

> **Type** str

max_comment_count

The maximum amount of comments this Artist can post.

> **Type** int

community_id

The ID of the community this Artist object was selected from.

> **Type** int

is_enabled
> If the Artist account is enabled.
>
> > **Type** bool

has_new_to_fans
> If the Artist has a new post for fans.
>
> > **Type** bool

has_new_private_to_fans
> If the Artist has a new private post for fans.
>
> > **Type** bool

to_fan_last_id
> The latest tofan post ID.
>
> > **Type** int

to_fan_last_created_at
> When the artist's last tofan post was created.

to_fan_last_expire_in
> When the artist's last tofan post expires.

birthday_img_url
> A direct image url to the artist's birthday image.
>
> > **Type** str

community
> The community the Artist is in.
>
> > **Type** *Community*

posts
> A list of posts the Artist has.
>
> > **Type** list

3.5 Comment

class Weverse.models.**Comment**(***kwargs*)
> A Comment object that represents a Weverse Comment that belongs to an Artist.
>
> It is not suggested to create a Comment manually, but rather through the following method: `Weverse.objects.create_comment_objects`
>
> The information retrieved on a Comment is directly from the Weverse API and altered to fit this class.
>
> > **Parameters**
> >
> > - **id** (`int`) – The ID of the comment.
> >
> > - **body** (`str`) – The comment content AKA the body of the message.
> >
> > - **comment_count** (`int`) – Amount of comments inside of this comment (replies).
> >
> > - **like_count** (`int`) – Amount of likes on the comment.
> >
> > - **has_my_like** (`bool`) – Whether the client has liked the comment.

- **is_blind** (bool) – NOT SURE WHAT THIS IS

- **post_id** (int) – The Post ID that the comment was created under.

- **created_at** – The time the comment was created.

- **updated_at** – The time the comment was updated.

id

The ID of the comment.

> **Type** int

body

The comment content AKA the body of the message.

> **Type** str

comment_count

Amount of comments inside of this comment (replies).

> **Type** int

like_count

Amount of likes on the comment.

> **Type** int

has_my_like

Whether the client has liked the comment.

> **Type** bool

is_blind

NOT SURE WHAT THIS IS

> **Type** bool

post_id

The Post ID that the comment was created under.

> **Type** int

created_at

The time the comment was created.

updated_at

The time the comment was updated.

post

The Post Object the comment belongs to.

> **Type** *Post*

3.6 Post

class Weverse.models.**Post**(***kwargs*)

A Post object that represents a Weverse Post.

It is not suggested to create a Post manually, but rather through the following method: *Weverse.objects. create_post_objects*

The information retrieved on a Post is directly from the Weverse API and altered to fit this class.

> **Parameters**
>
> - **id** (*int*) – The ID of the post.
> - **community_tab_id** (*int*) – The tab the post is under.
> - **type** (*str*) – The type of Post.
> - **body** (*str*) – Body Message on the Post.
> - **comment_count** (*int*) – Current amount of comments on the Post
> - **like_count** (*int*) – Current amount of likes on the Post
> - **max_comment_count** (*int*) – Maximum amount of comments that can be on the Post
> - **has_my_like** (*bool*) – If the client user has the post liked.
> - **has_my_bookmark** (*bool*) – If the client user has the post bookmarked.
> - **created_at** – When the post was created
> - **updated_at** – When the post was last modified.
> - **is_locked** (*bool*) – Whether the post is locked.
> - **is_blind** (*bool*) – Whether the post is visible?? Unknown
> - **is_active** (*bool*) – Whether the post is active.
> - **is_private** (*bool*) – Whether the post is private.
> - **photos** (List[*Photo*]) – A list of photos under the post.
> - **is_hot_trending_post** (*bool*) – If the post is trending.
> - **is_limit_comment** (*bool*) – If the comments are limited.
> - **artist_comments** (List[*Comment*]) – The Artist comments under the post.
> - **community_artist_id** (*int*) – The Community Artist ID that made the post.
> - **artist_id** (*int*) – The ID of the Artist that made the post.

> **id**
>
> The ID of the post.
>
> > **Type** int

> **community_tab_id**
>
> The tab the post is under.
>
> > **Type** int

> **type**
>
> The type of Post.
>
> > **Type** str

body
> Body Message on the Post.
>
> > **Type** str

comment_count
> Current amount of comments on the Post
>
> > **Type** int

like_count
> Current amount of likes on the Post
>
> > **Type** int

max_comment_count
> Maximum amount of comments that can be on the Post
>
> > **Type** int

has_my_like
> If the client user has the post liked.
>
> > **Type** bool

has_my_bookmark
> If the client user has the post bookmarked.
>
> > **Type** bool

created_at
> When the post was created

updated_at
> When the post was last modified.

is_locked
> Whether the post is locked.
>
> > **Type** bool

is_blind
> Whether the post is visible?? Unknown
>
> > **Type** bool

is_active
> Whether the post is active.
>
> > **Type** bool

is_private
> Whether the post is private.
>
> > **Type** bool

photos
> A list of photos under the post.
>
> > **Type** List[*Photo*]

is_hot_trending_post
> If the post is trending.
>
> > **Type** bool

is_limit_comment
> If the comments are limited.
>
> > **Type** bool

artist_comments
> The Artist comments under the post.
>
> > **Type** List[*Comment*]

community_artist_id
> The Community Artist ID that made the post.
>
> > **Type** int

artist_id
> The ID of the Artist that made the post.
>
> > **Type** int

artist
> The Artist Object the post belongs to.
>
> > **Type** *Artist*

3.7 Tab

class Weverse.models.**Tab**(*tab_id=None*, *name=None*)
> A Post object that represents a Weverse Post.
>
> It is not suggested to create a Post manually, but rather through the following method: `Weverse.objects.create_post_objects`
>
> The information retrieved on a Post is directly from the Weverse API and altered to fit this class.
>
> > **Parameters**
> >
> > - **tab_id** (`[Optional] int`) – The ID of the Tab.
> > - **name** (`[Optional] str`) – The Tab name.
>
> **id**
> > The ID of the Tab.
> >
> > > **Type** int
>
> **name**
> > The Tab name.
> >
> > > **Type** str

3.8 Media

class Weverse.models.**Media**(***kwargs*)

A Media object that represents a Weverse Media Post.

It is not suggested to create a Media object manually, but rather through the following method: *Weverse. objects.create_media_object*

The information retrieved on Media is directly from the Weverse API and altered to fit this class.

> **Parameters**
>
> - **id** (*int*) – ID of the Media post.
> - **community_id** (*int*) – ID of the Community the media post was made in.
> - **body** (*str*) – The media content AKA the body of the message.
> - **type** (*str*) – The type of media post it is.
> - **thumbnail_path** (*str*) – The (url??) of the thumbnail.
> - **title** (*str*) – The title of the media post.
> - **level** – The level of access the media post is categorized under.
> - **video_link** (*str*) – The video link supplied under the media post.
> - **youtube_id** (*str*) – The youtube video ID.

id

ID of the Media post.

> **Type** int

community_id

ID of the Community the media post was made in.

> **Type** int

body

The media content AKA the body of the message.

> **Type** str

type

The type of media post it is.

> **Type** str

thumbnail_path

The (url??) of the thumbnail.

> **Type** str

title

The title of the media post.

> **Type** str

level

The level of access the media post is categorized under.

video_link

The video link supplied under the media post.

> **Type** str

youtube_id
    The youtube video ID.

        Type str

MODEL CREATION

Weverse.objects.**create_artist_objects**(*current_artists: list*) → list
>   Creates artist objects based on a list of information sent in and returns the objects.

>>   **Parameters** **current_artists** – Artist information received from endpoint.

>>   **Returns** List[*Artist*]

Weverse.objects.**create_comment_objects**(*current_comments: list*) → list
>   Creates & Returns comment objects based on a list of comments

>>   **Parameters** **current_comments** – comment information from endpoint.

>>   **Returns** List[*Comment*]

Weverse.objects.**create_community_objects**(*current_communities: list*) → dict
>   Creates community objects based on a list of information sent in and returns the objects.

>>   **Parameters** **current_communities** – Community information received from endpoint.

>>   **Returns** dict{community id: *Community*}

Weverse.objects.**create_media_object**(*media_info: dict*) → *Weverse.models.media.Media*
>   Creates and returns a media object

>>   **Parameters** **media_info** – media information from endpoint.

>>   **Returns** *Media*

Weverse.objects.**create_notification_objects**(*current_notifications: list*) → list
>   Creates notification objects based on a list of information sent in and returns the objects.

>>   **Parameters** **current_notifications** – Notification information received from endpoint.

>>   **Returns** List[*Notification*]

Weverse.objects.**create_photo_objects**(*current_photos: list*) → list
>   Creates & Returns photo objects based on a list of photos

>>   **Parameters** **current_photos** – photo information from endpoint.

>>   **Returns** List[*Photo*]

Weverse.objects.**create_post_objects**(*current_posts: list*, *community:*
Weverse.models.community.Community, *new=False*) → list
>   Creates post objects based on a list of posts sent in and the community and returns the objects.

>>   **Parameters**

>>>   • **current_posts** – Post information received from endpoint.

>>>   • **community** – *Community* that the post belongs in.

- **new** – bool Whether or not the post is new.

> **Returns** List[*Post*]

Weverse.objects.**create_tab_objects**(*current_tabs: list*) → list
    Creates tab objects based on a list of information sent in and returns the objects.

> **Parameters** **current_tabs** – Tab information received from endpoint.

> **Returns** List[*Tab*]

EXCEPTIONS

5.1 Invalid Token

exception Weverse.**InvalidToken**
> An Exception Raised When an Invalid Token was Supplied.

5.2 Page Not Found

exception Weverse.**PageNotFound**(*url*)
> An Exception Raised When a link was not found.
>
>> **Parameters url** (str) – The link that was not found.

5.3 Being Rate Limited

exception Weverse.**BeingRateLimited**
> An Exception Raised When Weverse Is Being Rate-Limited.

SIX

GET ACCOUNT TOKEN

Your account token is needed (Will need to be updated about every 6 months iirc).

In order to get your account token, go to https://www.weverse.io/ and Inspect Element (F12).

Then go to the *Network* tab and filter by *XHR*.

Then refresh your page (F5) and look for *info* or *me* under *XHR*.

Under Headers, scroll to the bottom and view the request headers.

You want to copy everything past *authorization: Bearer*.

For example, you may see (This is just an example):

`authorization:  Bearer ABCDEFGHIJKLMNOPQRSTUVWXYZ`

Then `ABCDEFGHIJKLMNOPQRSTUVWXYZ` would be your auth token for Weverse.

It is suggested to have the auth token as an environment variable.

SEVEN

ASYNCHRONOUS USAGE

```python
# Asynchronous
import asyncio
import aiohttp
from Weverse.error import InvalidToken
from Weverse.weverseasync import WeverseClientAsync


token = "fake_token"  # REQUIRED
# It is advised to pass in your own web session as it is not closed in Weverse
web_session = aiohttp.ClientSession()
weverse_client = WeverseClientAsync(authorization=token, verbose=True, loop=asyncio.get_
→event_loop(), web_session=web_session)
try:
    await weverse_client.start()  # creates all the cache needed for your account.
except InvalidToken:
    print("Invalid Token")
```

SYNCHRONOUS USAGE

```python
# Synchronous
import requests
from Weverse.weversesync import WeverseClientSync
from Weverse.error import InvalidToken


token = "fake_token"  # REQUIRED
# It is advised to pass in your own web session as it is not closed in Weverse
web_session = requests.Session()  # A session is created by default
weverse_client = WeverseClientSync(authorization=token, verbose=True)
try:
    weverse_client.start()  # creates all the cache needed for your account.
except InvalidToken:
    print("Invalid Token")
```

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

W