

---

**Weverse**

**MujyKun**

**Jun 27, 2022**



## CONTENTS:

<b>1</b>	<b>WeverseClient</b>	<b>1</b>
<b>2</b>	<b>Clients</b>	<b>7</b>
2.1	WeverseClientSync . . . . .	7
2.2	WeverseClientAsync . . . . .	9
<b>3</b>	<b>Models</b>	<b>15</b>
3.1	Community . . . . .	15
3.2	Notification . . . . .	17
3.3	Photo . . . . .	19
3.4	Video . . . . .	21
3.5	VideoStream . . . . .	22
3.6	Artist . . . . .	22
3.7	Comment . . . . .	25
3.8	Post . . . . .	26
3.9	Tab . . . . .	30
3.10	Media . . . . .	30
3.11	Announcement . . . . .	32
<b>4</b>	<b>Model Creation</b>	<b>35</b>
<b>5</b>	<b>Exceptions</b>	<b>39</b>
5.1	Invalid Token . . . . .	39
5.2	Page Not Found . . . . .	39
5.3	Being Rate Limited . . . . .	39
<b>6</b>	<b>Get Account Token</b>	<b>41</b>
<b>7</b>	<b>Asynchronous Usage</b>	<b>43</b>
<b>8</b>	<b>Synchronous Usage</b>	<b>45</b>
<b>9</b>	<b>Indices and tables</b>	<b>47</b>
	<b>Python Module Index</b>	<b>49</b>
	<b>Index</b>	<b>51</b>



## 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.
- **authorization** (*str*) – The account token to connect to the Weverse API. In order to find your token, please refer to [Get Account Token](#)
- **username** (*str*) – The email or username associated with the account.
- **password** (*str*) – The password associated with the account.
- **hook** – A passed in method that will be called every time there is a new notification. This method must take in a list of [models.Notification](#) objects.

### verbose

Whether to print out verbose messages.

#### Type

bool

### web\_session

An aiohttp or requests client session.

### user\_notifications

Most recent notifications of the account connected.

#### Type

list

### cache\_loaded

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

#### Type

bool

### user\_endpoint

User info endpoint.

**Type**  
str

**all\_posts**

All posts in cache where the Post ID is the key and the value is the Post Object

**Type**  
dict(*Post*)

**all\_artists**

All artists in cache where the Artist ID is the key and the value is the Artist Object

**Type**  
dict(*Artist*)

**all\_comments**

All comments in cache where the Comment ID is the key and the value is the Comment Object

**Type**  
dict(*Comment*)

**all\_notifications**

All notifications in cache where the Notification ID is the key and the value is the Notification Object

**Type**  
dict(*Notification*)

**all\_photos**

All photos in cache where the Photo ID is the key and the value is the Photo Object

**Type**  
dict(*Photo*)

**all\_communities**

All communities in cache where the Community ID is the key and the value is the Community Object

**Type**  
dict(*Community*)

**all\_media**

All media in cache where the Media ID is the key and the value is the Media Object

**Type**  
dict(*Media*)

**all\_tabs**

All tabs in cache where the Tab ID is the key and the value is the Tab Object

**Type**  
dict(*Tab*)

**all\_videos**

All videos in cache where the Video URL is the key and the value is the Video Object

**Type**  
dict(*Video*)

**all\_announcements**

All announcements/notices in cache where the Announcement ID is the key and the value is the Announcement Object

**Type**dict(*Announcement*)**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: Union[Notification, str]*) → str

Determine the post type based on the notification body or the Notification object itself.

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

**Parameters****notification** – The message body of the notification or the notification itself.**Returns**

A string with either “comment”, “media”, “post”, or “announcement”.

**get\_announcement\_by\_id**(*announcement\_id*) → Optional[*Announcement*]

Get Announcement by the ID

**Parameters****announcement\_id** – Media ID**Returns**Optional[*Announcement*]**get\_artist\_by\_id**(*artist\_id*) → Optional[*Artist*]

Get artist by their ID.

**Parameters****artist\_id** – The artist’s ID**Returns**Optional[*Artist*]**get\_comment\_by\_id**(*comment\_id*) → Optional[*Comment*]Get a comment by the ID :param comment\_id: Comment ID :returns: Optional[*Comment*]**get\_community\_by\_id**(*community\_id*) → Optional[*Community*]

Get a community by the ID.

**Parameters****community\_id** – Community ID**Returns**Optional[*Community*]**get\_media\_by\_id**(*media\_id*) → Optional[*Media*]

Get Media by the ID

**Parameters****media\_id** – Media ID**Returns**Optional[*Media*]**get\_new\_notifications()** → List[*Notification*]

Will get the new notifications from the last notification check.

Should only be used after check\_new\_user\_notifications OR update\_cache\_from\_notification.

**Returns**List[*Notification*]**get\_notification\_by\_id(notification\_id)** → Optional[*Notification*]

Get a notification by the ID

**Parameters****notification\_id** – Notification ID**Returns**Optional[*Notification*]**get\_photo\_by\_id(photo\_id)** → Optional[*Photo*]

Get a photo by the ID

**Parameters****photo\_id** – Photo ID**Returns**Optional[*Photo*]**get\_post\_by\_id(post\_id)** → Optional[*Post*]

Get a post by the ID

**Parameters****post\_id** – Post ID**Returns**Optional[*Post*]**get\_tab\_by\_id(tab\_id)** → Optional[*Tab*]

Get tab by their ID.

**Parameters****tab\_id** – The tab ID**Returns**Optional[*Tab*]**get\_video\_by\_url(video\_url)** → Optional[*Video*]

Get a video by the direct URL.

**Parameters****video\_url** – URL of the video**Returns**Optional[*Video*]**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.

**property public\_weverse\_key: str**

Hard-coded weverse key

**stop()**

Stop the hook loop.



## 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.

This endpoint has been acting a bit off and not producing accurate results. It would be recommended to instantly get new notifications with `update_cache_from_notification` instead.

**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\_media**(community: *Community*)

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

**Parameters**

**community** – *Community* the posts exist under.

**create\_post**(community: *Community*, post\_id) → *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*: [Community](#), *next\_page\_id*: *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\_announcement**(*community\_id*: *int*, *announcement\_id*: *int*) → Optional[[Announcement](#)]

Receive announcement object based on announcement id.

**Parameters**

- **community\_id** – The ID of the community the media belongs to.
- **announcement\_id** – The ID of the announcement to fetch.

**Returns**

[Announcement](#) or *NoneType*

**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**(*create\_old\_posts=False, create\_notifications=True, create\_media=False*)

Creates internal cache.

This is the main process that should be run.

#### Parameters

- **create\_old\_posts** – (bool) Whether to create cache for old posts.
- **create\_notifications** – (bool) Whether to create/update cache for old notifications.
- **create\_media** – (bool) Whether to create/update cache for old media.

#### Raises

*Weverse.error.InvalidToken* If the token was invalid.

#### Raises

*Weverse.error.BeingRateLimited* If the client is being rate-limited.

#### Raises

*Weverse.error.LoginFailed* Login process had failed.

#### Raises

*Weverse.error.InvalidCredentials* If the user credentials were invalid or not provided.

**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**() → List[*Notification*]

Grab a new post based from new notifications and add it to cache.

Will also return the new notifications found.

#### Returns

List[*models.Notification*]

## 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 *WeverseClient*.

**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.

This endpoint has been acting a bit off and not producing accurate results. It would be recommended to instantly get new notifications with `update_cache_from_notification` instead.

**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.

**property cookies:** Optional[dict]

Get the user's cookies in order to access media.

**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**(*specific\_community\_ids: List[int] = None*)

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

**Parameters**

**specific\_community\_ids** – List[int] Will only do this list of community ids from the already existing communities.

This is a coroutine and must be awaited.

**async create\_media**(*community: Community*)

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

**Parameters**

**community** – *Community* the posts exist under.

**async create\_post**(*community: Community, post\_id*) → *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: Community, next\_page\_id: 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 download\_video\_stream**(*video\_stream\_obj*: *VideoStream*, *output\_file\_path*)

Download a video stream to a local folder.

#### Parameters

- **video\_stream\_obj** (*VideoStream*) –
- **output\_file\_path** (*str*) – Full file path with file extension.

#### Return type

Returns False if no community id is found with the *VideoStream* object.

**async fetch\_announcement**(*community\_id*: *int*, *announcement\_id*: *int*) → Optional[*Announcement*]

Receive announcement object based on announcement id.

This is a coroutine and must be awaited.

#### Parameters

- **community\_id** – The ID of the community the media belongs to.
- **announcement\_id** – The ID of the announcement to fetch.

#### Returns

*Announcement* or *NoneType*

**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*) → *str*

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*) → Optional[*Media*]

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 follow\_all\_communities()**

Follow all communities on Weverse

**async follow\_community**(*community\_id: Union[int, str], attempts: int = 0*)

Follow a community

**Parameters**

- **community\_id** – Union[int, str] The community ID to follow.
- **attempts** – int The number of attempts for choosing a nickname after error.

**async get\_all\_community\_ids()** → List[int]

Get all the communities on Weverse.

**Returns**

List[int] A list of community ids

**async get\_cookies**(*video\_url\_without\_drm\_type*) → Optional[dict]

Get the user's cookies in order to access media.

**Parameters**

**video\_url\_without\_drm\_type** – EX: <https://weversewebapi.weverse.io/wapi/v1/communities/2/videos/4093>

**Returns**

Optional[dict] A dictionary containing a signed cookie.

**async get\_user\_notifications()**

Get a list of updated user notification objects.

This is a coroutine and must be awaited.

**Returns**

List[*Notification*]

**async run\_blocking\_code**(*funcs, \*args, \*\*kwargs*) → list

Run blocking code safely in a new thread. DO NOT pass in an asynchronous function. If an asynchronous function has blocking code, the event loop will also block. There were several attempts made to make it compatible with asynchronous functions, but it was a headache to work with.

**Parameters**

- **funcs** – The blocking function that needs to be called. May also pass in a list of functions with the 0th index as the callable function, the 1st index as the args for that function, and the 2nd index as the kwargs for that function.
- **args** – The args to pass into the blocking function.
- **kwargs** – The keyword args to pass into the blocking function.

**Returns**

List of results in no particular order. Make sure the output can be managed with no specific order.



**async start**(*create\_old\_posts=False, create\_notifications=True, create\_media=False, follow\_new\_communities=True*)

Creates internal cache.

This is the main process that should be run.

This is a coroutine and must be awaited.

#### Parameters

- **create\_old\_posts** – (bool) Whether to create cache for old posts.
- **create\_notifications** – (bool) Whether to create/update cache for old notifications.
- **create\_media** – (bool) Whether to create/update cache for old media.
- **follow\_new\_communities** – bool Check for new communities and automatically follow them.

#### Raises

*Weverse.error.InvalidToken* If the token was invalid.

#### Raises

*Weverse.error.BeingRateLimited* If the client is being rate-limited.

#### Raises

*Weverse.error.LoginFailed* Login process had failed.

#### Raises

*asyncio.exceptions.TimeoutError* Waited too long for a login.

#### Raises

*Weverse.error.InvalidCredentials* If the user credentials were invalid or not provided.

**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**() → List[*Notification*]

Grab a new post based from new notifications and add it to cache.

Will also return the new notifications found.

This is a coroutine and must be awaited.

#### Returns

List[*models.Notification*]



## 3.1 Community

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

A Community object that represents a Weverse Community.

It is not suggested to create a Community manually, but rather through the following method: [Weverse.objects.create\\_community\\_objects](#)

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

**x == y**

Checks if two Communities have the same ID.

**x != y**

Checks if two Communities do not have the same ID.

**str(x)**

Returns the Community's name.

### 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.

**x == y**

Checks if two Notifications have the same ID.

**x != y**

Checks if two Notifications do not have the same ID.

**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.

**x == y**

Checks if two Photo objects have the same ID.

**x != y**

Checks if two Photo objects do not have the same ID.

**str(x)**

Returns the file name.

**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

**media\_id**

The media ID of the photo (if there is one).

**Type**

Optional[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 Video

**class** Weverse.models.Video(\*\*kwargs)

A Video object that represents a Weverse Video that belongs to media or a post.

It is not suggested to create a Video manually, but rather through the following method: [Weverse.objects.create\\_video\\_objects](#)

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

Videos do not have unique IDs.

**x == y**

Check if the Video URL and Post are the same.

**x != y**

Check if the Video objects are not equal.

**str(x)**

Returns the Video URL.

**len(x)**

Returns the length of the video in seconds.

### Parameters

- **video\_url** (*int*) – Direct URL to the video.
- **thumbnail\_url** (*str*) – URL of the thumbnail.
- **thumbnail\_width** (*int*) – Width of the thumbnail
- **thumbnail\_height** (*int*) – Height of the thumbnail.
- **length** (*int*) – Duration of the video in seconds.

**video\_url**

Direct URL to the video.

**Type**

int

**thumbnail\_url**

URL of the thumbnail.

**Type**

str

**thumbnail\_width**

Width of the thumbnail

**Type**

int

**thumbnail\_height**

Height of the thumbnail.

**Type**

int

**playtime**

Duration of the video in seconds.

**Type**

int

**post**

The Post Object the video belongs to.

**Type**

Optional[[Post](#)]

## 3.5 VideoStream

```
..autoclass:: Weverse.models.VideoStream
```

members

## 3.6 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.

```
x == y
```

Checks if two Artists have the same ID.

```
x != y
```

Checks if two Artists do not have the same ID.

```
str(x)
```

Returns the Artist's primary name.

**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

## 3.7 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.

**x == y**

Checks if two Comments have the same ID.

**x != y**

Checks if two Comments do not have the same ID.

**str(x)**

Returns the comment's body.

**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*

## 3.8 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.

**x == y**

Checks if two Post objects have the same ID.

**x != y**

Checks if two Post objects do not have the same ID.

**str(x)**

Returns the Post body message.

**len(x)**

Returns the amount of images (not videos) available.

#### 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.
- **videos** (List[*Video*]) – A list of videos 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*]

**videos**

A list of videos under the post.

**Type**

List[*Video*]

**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*

## 3.9 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.

**x == y**

Checks if two Tab objects have the same ID.

**x != y**

Check if the IDs of the Tab objects are not equal.

**str(x)**

Returns the Tab name.

**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

## 3.10 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.

**x == y**

Checks if two Media objects have the same ID.

**x != y**

Checks if two Media objects do not have the same ID.

**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

**photos**

A list of photos under the media post.

**Type**

List[*Photo*]

**videos**

A list of videos under the media post.

**Type**

List[*Video*]

## 3.11 Announcement

**class** Weverse.models.**Announcement**(\*\*kwargs)

An Announcement object that represents a Weverse Notice for a Community.

It is not suggested to create an Announcement manually, but rather through the following method: `Weverse.objects.create_announcement_objects`

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

**x == y**

Checks if two Announcement objects have the same ID.

**x != y**

Checks if two Announcement objects do not have the same ID.

**str(x)**

Returns the Announcement content.

**Parameters**

- **id** (*int*) – The ID of the post.
- **communityId** (*int*) – The Community ID.
- **title** (*str*) – The title of the announcement notice.
- **content** (*str*) – The HTML body of the page notice.
- **createdAt** (*str*) – Timestamp with the date of when the announcement was created.
- **exposedAt** (*str*) – Timestamp with the date of when the announcement was released.
- **categoryId** (*int*) – Category that the announcement belongs to (used for paginating or quick endpoint access)

- **fcOnly** (*bool*) – If only premium members have access to the announcement.

**id**

The ID of the post.

**Type**

int

**community\_id**

The Community ID.

**Type**

int

**title**

The title of the announcement notice.

**Type**

str

**html\_content**

The HTML body of the page notice.

**Type**

str

**created\_at**

Timestamp with the date of when the announcement was created.

**Type**

str

**exposed\_at**

Timestamp with the date of when the announcement was released.

**Type**

str

**category\_id**

Category that the announcement belongs to (used for paginating or quick endpoint access)

**Type**

int

**fc\_only**

If only premium members have access to the announcement.

**Type**

bool

**image\_url**

An image url if one is present.

**Type**

Optional[str]

**content**

Body Content without the HTML tags.

**Type**

str



## MODEL CREATION

`Weverse.objects.create_announcement_object(announcement_info: dict) → Announcement`

Creates and returns an announcement object

**Parameters**

**announcement\_info** – Announcement information from endpoint.

**Returns**

*Announcement*

`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, already_existing: Optional[Dict[int, Community]] = None) → dict`

Creates community objects based on a list of information sent in and returns the objects.

**Parameters**

- **current\_communities** – A list of communities from the endpoint being followed. Community information received from endpoint.
- **already\_existing** – List[*Community*] Already existing Communities that should not be replaced.

**Returns**

dict{community id: *Community*}

`Weverse.objects.create_media_object(media_info: dict, ignore_photos=False, ignore_videos=False) → Media`

Creates and returns a media object

**Parameters**

- **media\_info** – media information from endpoint.
- **ignore\_photos** – Whether to ignore the photos that belong in the media object. (Other methods can create it themselves.)
- **ignore\_videos** – Whether to ignore the videos that belong in the media object. (Other methods can create it themselves.)

**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: 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]*

`Weverse.objects.create_video_objects(current_videos: list, community_id=None) → list`

Creates & Returns video objects based on a list of videos.

**Parameters**

- **current\_videos** – Video information from api endpoint.
- **community\_id** – Community ID

**Returns***List[Video]*



`Weverse.objects.iterate_community_media_categories(all_media_categories: dict) →`  
`[List[Weverse.models.media.Media], List[dict]]`

Iterates through community media categories, creates Media posts and returns a list of them.

**Parameters**

**all\_media\_categories** – A dict containing media posts that are filtered by category.

**Returns**

[List[[Media](#)], List[dict]] A list of Video Media objects and a list of dicts containing photo media objects to later make own calls on to retrieve photos.



## 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.



## GET ACCOUNT TOKEN

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

Note that it is now possible to log-in with a username and password to prevent manual updates.

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.

**IMPORTANT NOTE:** Not all korean key-phrases may be logged. Scroll to the bottom of the Weverse page when you are logged in and click “English” to set the account language to English.



## ASYNCHRONOUS USAGE

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

# THERE IS A MORE DETAILED EXAMPLE IN THE EXAMPLES FOLDER
# https://github.com/MujoyKun/Weverse/blob/main/examples/asynchronous.py

token = "fake_token" # REQUIRED
# THE EXAMPLE IN THE EXAMPLES FOLDER WILL SHOW YOU HOW TO LOGIN WITH A USERNAME AND
# ↪ PASSWORD AND SET UP HOOKS.

# It is advised to pass in your own web session as it is not closed in Weverse
web_session = aiohttp.ClientSession() # A session is created by default
weverse_client = WeverseClientAsync(authorization=token, verbose=True, loop=asyncio.get_
# ↪ event_loop(),
                                web_session=web_session)

try:
    # creates all the cache that is specified. If the create parameters are set to True,
    # ↪ they will take a very long time.
    await weverse_client.start(create_old_posts=True, create_media=True)
except InvalidToken:
    print("Invalid Token")
```





## SYNCHRONOUS USAGE

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

# THERE IS A MORE DETAILED EXAMPLE IN THE EXAMPLES FOLDER
# https://github.com/MujyKun/Weverse/blob/main/examples/synchronous.py

token = "fake_token" # REQUIRED
# THE EXAMPLE IN THE EXAMPLES FOLDER WILL SHOW YOU HOW TO LOGIN WITH A USERNAME AND
# ↪ PASSWORD AND SET UP HOOKS.

# 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:
    # creates all the cache that is specified. If the create parameters are set to True,
    # ↪ they will take a very long time.
    weverse_client.start(create_old_posts=True, create_media=True)
except InvalidToken:
    print("Invalid Token")
```



## INDICES AND TABLES

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



## PYTHON MODULE INDEX

### W

`Weverse.objects`, [35](#)



## INDEX

### A

`all_announcements` (*Weverse.WeverseClient* attribute), 2  
`all_artists` (*Weverse.WeverseClient* attribute), 2  
`all_comments` (*Weverse.WeverseClient* attribute), 2  
`all_communities` (*Weverse.WeverseClient* attribute), 2  
`all_media` (*Weverse.WeverseClient* attribute), 2  
`all_notifications` (*Weverse.WeverseClient* attribute), 2  
`all_photos` (*Weverse.WeverseClient* attribute), 2  
`all_posts` (*Weverse.WeverseClient* attribute), 2  
`all_tabs` (*Weverse.WeverseClient* attribute), 2  
`all_videos` (*Weverse.WeverseClient* attribute), 2  
`Announcement` (class in *Weverse.models*), 32  
`Artist` (class in *Weverse.models*), 22  
`artist` (*Weverse.models.Post* attribute), 29  
`artist_comments` (*Weverse.models.Post* attribute), 29  
`artist_id` (*Weverse.models.Notification* attribute), 18  
`artist_id` (*Weverse.models.Post* attribute), 29  
`artists` (*Weverse.models.Community* attribute), 16

### B

`Banner` (*Weverse.models.Community* attribute), 16  
`BeingRateLimited`, 39  
`birthday_img_url` (*Weverse.models.Artist* attribute), 24  
`body` (*Weverse.models.Comment* attribute), 25  
`body` (*Weverse.models.Media* attribute), 31  
`body` (*Weverse.models.Post* attribute), 28  
`bold_element` (*Weverse.models.Notification* attribute), 18

### C

`cache_loaded` (*Weverse.WeverseClient* attribute), 1  
`category_id` (*Weverse.models.Announcement* attribute), 33  
`check_new_user_notifications()` (*Weverse.WeverseClientAsync* method), 10  
`check_new_user_notifications()` (*Weverse.WeverseClientSync* method), 7  
`check_status()` (*Weverse.WeverseClient* method), 3

`check_token_works()` (*Weverse.WeverseClientAsync* method), 10  
`check_token_works()` (*Weverse.WeverseClientSync* method), 7  
`Comment` (class in *Weverse.models*), 25  
`comment_count` (*Weverse.models.Comment* attribute), 26  
`comment_count` (*Weverse.models.Post* attribute), 28  
`Community` (class in *Weverse.models*), 15  
`community` (*Weverse.models.Artist* attribute), 24  
`community_artist_id` (*Weverse.models.Post* attribute), 29  
`community_id` (*Weverse.models.Announcement* attribute), 33  
`community_id` (*Weverse.models.Artist* attribute), 24  
`community_id` (*Weverse.models.Media* attribute), 31  
`community_id` (*Weverse.models.Notification* attribute), 18  
`community_name` (*Weverse.models.Notification* attribute), 18  
`community_tab_id` (*Weverse.models.Post* attribute), 27  
`community_user_id` (*Weverse.models.Artist* attribute), 23  
`content` (*Weverse.models.Announcement* attribute), 33  
`content_index` (*Weverse.models.Photo* attribute), 20  
`contents_id` (*Weverse.models.Notification* attribute), 18  
`contents_type` (*Weverse.models.Notification* attribute), 18  
`cookies` (*Weverse.WeverseClientAsync* property), 10  
`create_announcement_object()` (in module *Weverse.objects*), 35  
`create_artist_objects()` (in module *Weverse.objects*), 35  
`create_comment_objects()` (in module *Weverse.objects*), 35  
`create_communities()` (*Weverse.WeverseClientAsync* method), 10  
`create_communities()` (*Weverse.WeverseClientSync* method), 7  
`create_community_artists_and_tabs()` (*Weverse.WeverseClientAsync* method), 10  
`create_community_artists_and_tabs()` (*Weverse.WeverseClientSync* method), 7

[erse.WeverseClientSync method](#)), 7  
[create\\_community\\_objects\(\)](#) (in module [Weverse.objects](#)), 35  
[create\\_media\(\)](#) ([Weverse.WeverseClientAsync method](#)), 10  
[create\\_media\(\)](#) ([Weverse.WeverseClientSync method](#)), 7  
[create\\_media\\_object\(\)](#) (in module [Weverse.objects](#)), 35  
[create\\_notification\\_objects\(\)](#) (in module [Weverse.objects](#)), 36  
[create\\_photo\\_objects\(\)](#) (in module [Weverse.objects](#)), 36  
[create\\_post\(\)](#) ([Weverse.WeverseClientAsync method](#)), 10  
[create\\_post\(\)](#) ([Weverse.WeverseClientSync method](#)), 7  
[create\\_post\\_objects\(\)](#) (in module [Weverse.objects](#)), 36  
[create\\_posts\(\)](#) ([Weverse.WeverseClientAsync method](#)), 10  
[create\\_posts\(\)](#) ([Weverse.WeverseClientSync method](#)), 7  
[create\\_tab\\_objects\(\)](#) (in module [Weverse.objects](#)), 36  
[create\\_video\\_objects\(\)](#) (in module [Weverse.objects](#)), 36  
[created\\_at](#) ([Weverse.models.Announcement attribute](#)), 33  
[created\\_at](#) ([Weverse.models.Comment attribute](#)), 26  
[created\\_at](#) ([Weverse.models.Post attribute](#)), 28

## D

[description](#) ([Weverse.models.Community attribute](#)), 16  
[determine\\_notification\\_type\(\)](#) ([Weverse.WeverseClient static method](#)), 3  
[download\\_video\\_stream\(\)](#) ([Weverse.WeverseClientAsync method](#)), 11

## E

[exposed\\_at](#) ([Weverse.models.Announcement attribute](#)), 33

## F

[fc\\_member](#) ([Weverse.models.Community attribute](#)), 16  
[fc\\_only](#) ([Weverse.models.Announcement attribute](#)), 33  
[fetch\\_announcement\(\)](#) ([Weverse.WeverseClientAsync method](#)), 11  
[fetch\\_announcement\(\)](#) ([Weverse.WeverseClientSync method](#)), 8  
[fetch\\_artist\\_comments\(\)](#) ([Weverse.WeverseClientAsync method](#)), 11  
[fetch\\_artist\\_comments\(\)](#) ([Weverse.WeverseClientSync method](#)), 8

[fetch\\_comment\\_body\(\)](#) ([Weverse.WeverseClientAsync method](#)), 11  
[fetch\\_comment\\_body\(\)](#) ([Weverse.WeverseClientSync method](#)), 8  
[fetch\\_media\(\)](#) ([Weverse.WeverseClientAsync method](#)), 11  
[fetch\\_media\(\)](#) ([Weverse.WeverseClientSync method](#)), 8  
[file\\_name](#) ([Weverse.models.Photo attribute](#)), 20  
[follow\\_all\\_communities\(\)](#) ([Weverse.WeverseClientAsync method](#)), 12  
[follow\\_community\(\)](#) ([Weverse.WeverseClientAsync method](#)), 12  
[full\\_name](#) ([Weverse.models.Community attribute](#)), 16

## G

[get\\_all\\_community\\_ids\(\)](#) ([Weverse.WeverseClientAsync method](#)), 12  
[get\\_announcement\\_by\\_id\(\)](#) ([Weverse.WeverseClient method](#)), 3  
[get\\_artist\\_by\\_id\(\)](#) ([Weverse.WeverseClient method](#)), 3  
[get\\_comment\\_by\\_id\(\)](#) ([Weverse.WeverseClient method](#)), 3  
[get\\_community\\_by\\_id\(\)](#) ([Weverse.WeverseClient method](#)), 3  
[get\\_cookies\(\)](#) ([Weverse.WeverseClientAsync method](#)), 12  
[get\\_media\\_by\\_id\(\)](#) ([Weverse.WeverseClient method](#)), 3  
[get\\_new\\_notifications\(\)](#) ([Weverse.WeverseClient method](#)), 4  
[get\\_notification\\_by\\_id\(\)](#) ([Weverse.WeverseClient method](#)), 4  
[get\\_photo\\_by\\_id\(\)](#) ([Weverse.WeverseClient method](#)), 4  
[get\\_post\\_by\\_id\(\)](#) ([Weverse.WeverseClient method](#)), 4  
[get\\_tab\\_by\\_id\(\)](#) ([Weverse.WeverseClient method](#)), 4  
[get\\_user\\_notifications\(\)](#) ([Weverse.WeverseClientAsync method](#)), 12  
[get\\_user\\_notifications\(\)](#) ([Weverse.WeverseClientSync method](#)), 8  
[get\\_video\\_by\\_url\(\)](#) ([Weverse.WeverseClient method](#)), 4  
[group\\_name](#) ([Weverse.models.Artist attribute](#)), 24

## H

[has\\_my\\_bookmark](#) ([Weverse.models.Post attribute](#)), 28  
[has\\_my\\_like](#) ([Weverse.models.Comment attribute](#)), 26  
[has\\_my\\_like](#) ([Weverse.models.Post attribute](#)), 28  
[has\\_new\\_private\\_to\\_fans](#) ([Weverse.models.Artist attribute](#)), 24  
[has\\_new\\_to\\_fans](#) ([Weverse.models.Artist attribute](#)), 24  
[home\\_banner](#) ([Weverse.models.Community attribute](#)), 16



`html_content` (*Weverse.models.Announcement attribute*), 33

## I

`icon` (*Weverse.models.Community attribute*), 16  
`icon_image_url` (*Weverse.models.Notification attribute*), 18  
`id` (*Weverse.models.Announcement attribute*), 33  
`id` (*Weverse.models.Artist attribute*), 23  
`id` (*Weverse.models.Comment attribute*), 25  
`id` (*Weverse.models.Community attribute*), 15  
`id` (*Weverse.models.Media attribute*), 31  
`id` (*Weverse.models.Notification attribute*), 17  
`id` (*Weverse.models.Photo attribute*), 19  
`id` (*Weverse.models.Post attribute*), 27  
`id` (*Weverse.models.Tab attribute*), 30  
`image_url` (*Weverse.models.Announcement attribute*), 33  
`InvalidToken`, 39  
`is_active` (*Weverse.models.Post attribute*), 29  
`is_birthday` (*Weverse.models.Artist attribute*), 23  
`is_blind` (*Weverse.models.Comment attribute*), 26  
`is_blind` (*Weverse.models.Post attribute*), 28  
`is_enabled` (*Weverse.models.Artist attribute*), 24  
`is_hot_trending_post` (*Weverse.models.Post attribute*), 29  
`is_limit_comment` (*Weverse.models.Post attribute*), 29  
`is_locked` (*Weverse.models.Post attribute*), 28  
`is_membership_content` (*Weverse.models.Notification attribute*), 18  
`is_online` (*Weverse.models.Artist attribute*), 23  
`is_private` (*Weverse.models.Post attribute*), 29  
`is_web_only` (*Weverse.models.Notification attribute*), 18  
`iterate_community_media_categories()` (in module *Weverse.objects*), 36

## L

`level` (*Weverse.models.Media attribute*), 31  
`like_count` (*Weverse.models.Comment attribute*), 26  
`like_count` (*Weverse.models.Post attribute*), 28  
`list_name` (*Weverse.models.Artist attribute*), 23  
`loop` (*Weverse.WeverseClientAsync attribute*), 9

## M

`max_comment_count` (*Weverse.models.Artist attribute*), 24  
`max_comment_count` (*Weverse.models.Post attribute*), 28  
`Media` (class in *Weverse.models*), 30  
`media_id` (*Weverse.models.Photo attribute*), 19  
`member_count` (*Weverse.models.Community attribute*), 16  
`message` (*Weverse.models.Notification attribute*), 17  
`module`

*Weverse.objects*, 35

## N

`name` (*Weverse.models.Artist attribute*), 23  
`name` (*Weverse.models.Community attribute*), 15  
`name` (*Weverse.models.Tab attribute*), 30  
`Notification` (class in *Weverse.models*), 17  
`notified_at` (*Weverse.models.Notification attribute*), 18

## O

`original_img_height` (*Weverse.models.Photo attribute*), 20  
`original_img_url` (*Weverse.models.Photo attribute*), 20  
`original_img_width` (*Weverse.models.Photo attribute*), 20

## P

`PageNotFound`, 39  
`Photo` (class in *Weverse.models*), 19  
`photos` (*Weverse.models.Media attribute*), 32  
`photos` (*Weverse.models.Post attribute*), 29  
`platform` (*Weverse.models.Notification attribute*), 19  
`playtime` (*Weverse.models.Video attribute*), 21  
`Post` (class in *Weverse.models*), 26  
`post` (*Weverse.models.Comment attribute*), 26  
`post` (*Weverse.models.Photo attribute*), 20  
`post` (*Weverse.models.Video attribute*), 22  
`post_id` (*Weverse.models.Comment attribute*), 26  
`posts` (*Weverse.models.Artist attribute*), 25  
`process_community_artists_and_tabs()` (*Weverse.WeverseClient* static method), 4  
`profile_img_path` (*Weverse.models.Artist attribute*), 23  
`profile_nick_name` (*Weverse.models.Artist attribute*), 23  
`public_weverse_key` (*Weverse.WeverseClient* property), 5

## R

`run_blocking_code()` (*Weverse.WeverseClientAsync* method), 12

## S

`show_member_count` (*Weverse.models.Community attribute*), 16  
`start()` (*Weverse.WeverseClientAsync* method), 12  
`start()` (*Weverse.WeverseClientSync* method), 8  
`stop()` (*Weverse.WeverseClient* method), 5

## T

`Tab` (class in *Weverse.models*), 30  
`tabs` (*Weverse.models.Community attribute*), 16

`thumbnail_height` (*Weverse.models.Video* attribute), 21  
`thumbnail_image_url` (*Weverse.models.Notification* attribute), 18  
`thumbnail_img_height` (*Weverse.models.Photo* attribute), 20  
`thumbnail_img_url` (*Weverse.models.Photo* attribute), 20  
`thumbnail_img_width` (*Weverse.models.Photo* attribute), 20  
`thumbnail_path` (*Weverse.models.Media* attribute), 31  
`thumbnail_url` (*Weverse.models.Video* attribute), 21  
`thumbnail_width` (*Weverse.models.Video* attribute), 21  
`title` (*Weverse.models.Announcement* attribute), 33  
`title` (*Weverse.models.Media* attribute), 31  
`to_fan_last_created_at` (*Weverse.models.Artist* attribute), 24  
`to_fan_last_expire_in` (*Weverse.models.Artist* attribute), 24  
`to_fan_last_id` (*Weverse.models.Artist* attribute), 24  
`translate()` (*Weverse.WeverseClientAsync* method), 13  
`translate()` (*Weverse.WeverseClientSync* method), 9  
`type` (*Weverse.models.Media* attribute), 31  
`type` (*Weverse.models.Post* attribute), 28

## U

`update_cache_from_notification()` (*Weverse.WeverseClientAsync* method), 13  
`update_cache_from_notification()` (*Weverse.WeverseClientSync* method), 9  
`updated_at` (*Weverse.models.Comment* attribute), 26  
`updated_at` (*Weverse.models.Post* attribute), 28  
`user_endpoint` (*Weverse.WeverseClient* attribute), 1  
`user_notifications` (*Weverse.WeverseClient* attribute), 1

## V

`verbose` (*Weverse.WeverseClient* attribute), 1  
`Video` (class in *Weverse.models*), 21  
`video_link` (*Weverse.models.Media* attribute), 31  
`video_url` (*Weverse.models.Video* attribute), 21  
`videos` (*Weverse.models.Media* attribute), 32  
`videos` (*Weverse.models.Post* attribute), 29

## W

`web_session` (*Weverse.WeverseClient* attribute), 1  
`Weverse.objects`  
    module, 35  
`WeverseClient` (class in *Weverse*), 1  
`WeverseClientAsync` (class in *Weverse*), 9  
`WeverseClientSync` (class in *Weverse*), 7

## Y

`youtube_id` (*Weverse.models.Media* attribute), 32