meapi

Release 0.7.0

David Lev

Jul 19, 2023

CONTENTS

meapi is a Python3 library to identify, discover and get information about phone numbers, indicate and report spam, get and manage socials, profile management and much more.

To **get started**, read the Setup guide.

For a **complete documentation** of available functions, see the Reference.

>> *For more information about Me® - Click here.*

CHAPTER

ONE

INSTALLATION

- **Install using pip3:**

```
pip3 install -U meapi
```

- **Install from source:**

```
git clone https://github.com/david-lev/meapi.git
cd meapi && python3 setup.py install
```

CHAPTER

TWO

FEATURES

2.1 Searching:

- Search phone numbers
- Get full user profile: profile picture, birthday, location, platform, socials and more
- Spam indication and report

2.2 Social:

- Get user social networks: facebook, instagram, twitter, spotify and more
- See how people call you
- Get mutual contacts
- See who watched your profile
- See who deleted you from his contacts book
- Get, publish and manage comments
- Get users location
- Read app notifications

2.3 Settings:

- Change profile information
- Configure social settings
- Connect social networks (And get verified blue check)
- Upload contacts and calls history
- Block profiles and numbers
- Delete or suspend your account

CHAPTER

THREE

USAGE

```python
from meapi import Me

# Initialize the client in interactive mode:
me = Me(interactive_mode=True)

#  Get information about any phone number:
res = me.phone_search('+972545416627')
if res:
    print(res.name)

#  Get user full profile:
if res.user:
    user = res.user
    print(f"{user.name=}, {user.email=}, {user.slogan=} {user.profile_picture=}")
    profile = res.get_profile()
    print(f"{profile.date_of_birth=}, {profile.location_name=}, {profile.gender=},
↪{profile.device_type=}")

    #  Get social media accounts:
    for social in profile.social:
        if social:
            print(f"Social media ({social.name}): {social.profile_url}")
            for post in social.posts:
                print(f"Post from {post.posted_at}:\n{post.text_first}\n{post.text_
↪second}")

#  Watch, approve and like comments:
for comment in me.get_comments():
    print(f"Comment from {comment.author.name}: {comment.message}")
    if comment.status == 'waiting':
        comment.approve()


#  Change your profile details:
my_profile = me.get_my_profile()
my_profile.first_name = 'David'
my_profile.last_name = 'Lev'

#  Get your profile in vCard format:
with open('/home/david/Downloads/my_vcard.vcf', 'w') as f:
```

(continues on next page)

(continued from previous page)

```
    f.write(my_profile.as_vcard(dl_profile_picture=True))

#  See how people call you:
for group in me.get_groups(sorted_by='count'):
    print(f"People named you '{group.name}' {group.count} times")

#  who watched your profile:
for watcher in me.who_watched(incognito=True, sorted_by='last_view'):
    print(f"The user '{watcher.user.name}' watched you {watcher.count} times")

#  who deleted you:
for deleted in me.who_deleted():
    print(f"The user '{deleted.user.name}' deleted you at {deleted.created_at}")

#  And much much more...
```

For more usage examples, read the Examples page.

CHAPTER

FOUR

REQUIREMENTS

- Python 3.6 or higher - https://www.python.org

CHAPTER

FIVE

SETUP AND USAGE

See the Documentation for detailed instructions

CHAPTER

SIX

DISCLAIMER

This application is intended for educational purposes only. Any use in professional manner or to harm anyone or any organization doesn't relate to me and can be considered as illegal. Me name, its variations and the logo are registered trademarks of NFO LTD. I have nothing to do with the registered trademark. I'm also not responsible for blocked accounts or any other damage caused by the use of this library. it is always recommended to use virtual phone numbers for testing purposes.

CHAPTER

SEVEN

CONTENTS

7.1 Setup

7.1.1 Installation

- **Install using pip3:**

```
pip3 install -U meapi
```

- **Install from source:**

```
git clone https://github.com/david-lev/meapi.git
cd meapi && python3 setup.py install
```

7.1.2 Initialization

There are two ways to initialize the `Me` client. *the programmatic way* and *the interactive way*.

The programmatic way

Here you will need to provide the phone number and the activation code and also try to catch the exceptions that may occur.

- This method is more suitable for automation, for example if you want to interact with the client from a web server.

```
>>> from meapi import Me
>>> me = Me(phone_number=1234567890, activation_code='123456', interactive_
↪mode=False) # intractive_mode=False is the default value
>>> my_profile = me.get_my_profile() # start using the client
```

You may want to catch some exceptions that may occur during the initialization process:

```
>>> from meapi import Me
>>> try:
...     me = Me(phone_number=1234567890, activation_code='123456', interactive_
↪mode=False)
... except NewAccountException: # If this is a new number that is not already open an␣
↪account
```

(continues on next page)

(continued from previous page)

```
...     me = Me(phone_number=1234567890, new_account_details=NewAccountDetails(
...         first_name="Chandler",
...         last_name="Bing",
...         email="chandler.bing@friends.tv"
...     ))
>>> my_profile = me.get_my_profile() # start using the client
```

- See the exceptions that may occur in the initialization process of the *Me* class (in the `raises` section).

The interactive way

Here you will be prompted to choose the authentication method and then you will be prompted to enter the necessary information.

- This method is more suitable for self use, for testing and for thous who are not familiar with the library.

```
>>> from meapi import Me
>>> me = Me(interactive_mode=True)
Welcome to Meapi!
How do you want to login?"
    1. Unofficial method (phone number)
    2. Official method (access token)
Enter your choice (1-2): 1
Enter your phone number: 1234567890
To get access token you need to authorize yourself.
Enter your verification code (6 digits): 123456
>>> my_profile = me.get_my_profile() # start using the client
```

7.1.3 Authentication

No matter the initialization method you choose, you will need to authenticate yourself in order to use the client. There are two ways for authentication, the unofficial method and the official method.

Unofficial method

important notes:

- **This method is for educational purposes only and its use is at your own risk.** See disclaimer.
- In this method you are going to verify as a user of the app and get a token with access to all the actions that the app provides.
- After verification, if you are connected to another device, Chances are you will be disconnected.
- For app users there is an Rate-limit of about `350` phone searches and `500` profile views per day.
- **There are two forms of active authentication, which means that you can initialize the client with the authentication code that you will receive actively in one of the following forms:**
    1. **WhatsApp:** Go to WhatsApp and send any message to this number (+447427928793) to get a verification code.

2. **Telegram:** Go to Telegram (http://t.me/Meofficialbot?start=__iw__XXXXXX Replace the `XXXXX` with your phone number) and hit `/start` to get a verification code.

- After you get the code, you can initialize the client with the following code:

```
>>> from meapi import Me
>>> me = Me(phone_number=1234567890, activation_code='123456')
```

- If this is a new number that is not already open an account, you will be required to fill in some details like name and email in order to create an account, See Registration.

Official method

- **You can also use the official verification and verify directly with an access token.**
  Me has an official API which can be accessed by submitting a formal request at this link (Probably paid). I guess you get a API KEY with which you can get an access token similar to the app. But I do not know what the scope of this token are and whether it is possible to contact with it the same endpoints that the official app addresses.
- If anyone can shed light on the official authentication method, I would be happy if he would contact me. so that I could better support it and exclude or add certain functions.
- If you have an access token and you are interested in connecting with it - do the following:

```
from meapi import Me
me = Me(access_token='XXXXXXXXXX') # Enter your access token
```

Keep in mind that the access token is valid for 24 hours, and to meapi it is not possible to refresh it without the `phone_number` and `pwd_token`, So you will need to catch the *`ForbiddenRequest`* exception and reinitialize the client with a new access token.

```
from meapi import Me
from meapi.utils.exceptions import ForbiddenRequest

me = Me(access_token='XXXXXXXXXX')

try:
    do_something_with_me()
except ForbiddenRequest:
    access_token = your_own_implementation_of_getting_new_access_token()
    me = Me(access_token=access_token)
```

- Again, if you have any information about the process of getting an access token in official method, contact me and I will be happy to add it to the library.

7.1.4 Registration

Because the authentication in Meapi is active (you don't sit and wait for the code, but initializing the client with the phone number and the activation code you have already received actively), You can also initialize the client with the necessary information in advance, good for cases of creating a new account:

```
from meapi import Me
from meapi.models.others import NewAccountDetails

data = NewAccountDetails(
    first_name="Phoebe
    last_name="Buffay",
    email="reginaphalange@friends.tv"
)

me = Me(
    phone_number=972123456789,
    activation_code='123456',
    new_account_details=data
)
```

If you don't know if this is a new account, try except for *NewAccountException*:

```
from meapi import Me
from meapi.models.others import NewAccountDetails
from meapi.utils.exceptions import NewAccountException

try:
    me = Me(phone_number=972123456789, activation_code='123456')
except NewAccountException:
    data = NewAccountDetails( # Ask the user for the details
        first_name=input("Enter first name: "),
        last_name=input("Enter last name: "),
        email=input("Enter email: ") or None
    )
    # no need to for the activation code because the credentials are already saved
    me = Me(phone_number=972123456789, new_account_details=data)

# Continue using the client
```

7.1.5 Credentials

meapi, needs to store your credentials (access token, refresh token etc.) in order to be able to use them later on without the need to login again every time you want to use the API.

- The default credentials manager is the - *JsonCredentialsManager*, which saves the credentials in a json file (`meapi_credentials.json` by default).
- You can implement your own credentials manager by implementing the *CredentialsManager* interface. See Credentials Manager.
- If you choose to use in the default credentials manager, the config file will be created in the location from which the library was called.

- The config file `meapi_credentials.json` format is:

```
{
    "972123456789": {
        "access": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.XXXXX",
        "pwd_token": "XXXXX-XXXXX-XXXXX-XXXXX-XXXXXXXXXXXX",
        "refresh": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.XXXXX"
    }
}
```

- You can copy/move this file between projects. Just specify the path to the config file in when you initialize the Me class:

```
from meapi import Me
from meapi.credentials_managers.json_files import JsonCredentialsManager

cm = JsonCredentialsManager(config_file="/home/david/meapi_credentials.json")
me = Me(phone_number=123456789, credentials_manager=cm)
```

7.1.6 SMS or Call

Many ask me why there is no option to verify via SMS or call.

Well, This is because authentication via WhatsApp and Telegram, is action of the user who sends a message to the bot and receives the verification code, as opposed to a call or SMS that requires an external service (which of course costs money) to make the call or send the SMS.

This is why at the app level, `Me` Apps used a secret key to generates a time-based hashed session token.

For obvious reasons I can not provide the key, but if you have knowledge of extracting secrets from APKs, look for the key and export it in the environment variables with the key `ANTI_SESSION_BOT_KEY`. meapi will detect the presence of the environment and offer you to use authentication via SMS or call.

- Needless to say, the functionality is for educational purposes only.

7.2 Reference

class meapi.**Me**(*phone_number: Optional[Union[int, str]] = None*, *access_token: Optional[str] = None*, *activation_code: Optional[str] = None*, *credentials_manager: Optional[CredentialsManager] = None*, *new_account_details: Optional[NewAccountDetails] = None*, *interactive_mode: bool = False*, *session: Optional[Session] = None*)

The `Me` Client. Used to interact with MeAPI.

- Source code: GitHub
- See Setup to get started.
- See Client for all the available methods.

Examples to setting up the client:

```
>>> from meapi import Me
>>> me = Me(phone_number=972123456789, activation_code='123456') # Unofficial␣
↪method with pre-provided activation code.
```

(continues on next page)

(continued from previous page)

```
>>> me = Me(interactive_mode=True) # With interactive mode (prompt for missing data␣
↪instead of raising exceptions).
>>> me = Me(access_token='xxxxxxxxxxx') # Official method, access token is␣
↪required (saved in memory).
>>> me = Me(phone_number=972123456789, credentials_
↪manager=RedisCredentialsManager(redis_con)) # With custom credentials manager.
>>> me = Me(phone_number=972123456789, activation_code='123456', new_account_
↪details=NewAccountDetails(first_name='Chandler', last_name='Bing')) # New account␣
↪registration.
```

Parameters

- **interactive_mode** (bool) – If True, the client will prompt for missing data instead of raising exceptions (See Interactive mode). *Default:* False.
- **phone_number** (str | int | None) – International phone number format (Required on the Unofficial method). *Default:* None.
- **access_token** (str | None) – Official access token (Required on the Official method). *Default:* None
- **activation_code** (str | None) – Activation code (NeedActivationCode exception will be raised if not provided and interactive_mode is False). *Default:* None
- **credentials_manager** (*CredentialsManager* | None) – Credentials manager to use in order to store and manage the credentials (See Credentials Manager). *Default:* JsonCredentialsManager.
- **new_account_details** (*NewAccountDetails* | None) – Account details for new account registration without the need for a prompt (even if interactive_mode is True). *Default:* None.
- **session** (requests.Session | None) – requests Session object. Default: None.

Raises

- ***NotValidPhoneNumber*** – If phone_number is not valid.
- ***NotValidAccessToken*** – If access_token is not valid.
- ***NeedActivationCode*** – If the account is not activated and activation_code is not provided and interactive_mode is False.
- ***IncorrectActivationCode*** – If the activation_code is incorrect and interactive_mode is False.
- ***ActivationCodeExpired*** – If the activation_code is correct but expired (Request a new one) and interactive_mode is False.
- ***MaxValidateReached*** – If the activation_code is not correct and the max number of tries reached (Request a new one).
- **ValueError** – If both phone_number and access_token are provided.
- ***NewAccountException*** – If this is new account and new_account_details is not provided and interactive_mode is False.
- **TypeError** – If credentials_manager is not an instance of *CredentialsManager*.
- ***BlockedAccount*** – If the account is blocked (You need to contact Me support).

- *IncorrectPwdToken* – If the pwd_token provided by the credentials manager is broken (You need to re-login).
- *PhoneNumberDoesntExists* – If the phone_number never request an activation code.
- *BrokenCredentialsManager* – If the credentials_manager not providing the expected data.
- *ForbiddenRequest* – In the official method, if the access_token is not valid.
- *FrozenInstance* – If you try to change the value of an attribute.

7.2.1 Client

This page is about the client methods.

- All the methods here are bound to to a Me instance.

```
from meapi import Me

me = Me(972123456789) # initialize the client
me.phone_search(972987654321)
me.get_profile('fdsfs-fdsfs-fdsfs-fdsfs')
me.get_socials()
me.get_settings()
me.get_groups()
```

Search

Me.**phone_search**(*phone_number: Union[str, int]*) → Optional[*Contact*]

Get information on any phone number.

```
>>> res = me.phone_search(phone_number=972545416627)
>>> res.name
'David'
```

Parameters
phone_number (str | int) – International phone number format.

Raises
SearchPassedLimit – if you passed the limit (About 350 per day in the unofficial auth method).

Returns
Contact object or None if no user exists on the provided phone number.

Return type
Contact | None

Profile

Me.**get_profile**(*uuid: Union[str, Contact, User]*) → *Profile*

Get user's profile.

For Me users (those who have registered in the app) there is an account `UUID` obtained when receiving information about the phone number *phone_search()*. With it, you can get social information and perform social actions.

```
>>> p = me.get_profile(uuid='f7930d0f-c8ba-425b-8478-013968f30466')
>>> print(p.name, p.email, p.profile_picture, p.gender, p.date_of_birth, p.slogan)
```

Parameters
uuid (`str` | *Contact* | *User*) – The user's UUID as `str` or *Contact* or *User* objects.

Raises
ProfileViewPassedLimit – if you passed the limit (About `500` per day in the unofficial auth method).

Returns
Profile object.

Return type
Profile

Me.**get_my_profile**(*only_limited_data: bool = False*) → *Profile*

Get your profile information.

```
>>> p = me.get_my_profile()
>>> p.as_dict()
>>> p.as_vcard()
>>> p.name = 'Changed Name'
>>> p.email = 'john.doe@gmail.com'
```

Parameters
only_limited_data (`bool`) – `True` to get only limited data (not included in the rate limit). *Default:* `False`.

Returns
Profile object.

Return type
Profile

Me.**update_profile_details**(*first_name: Optional[str] = False, last_name: Optional[str] = False, email: Optional[str] = False, gender: Optional[str] = False, slogan: Optional[str] = False, profile_picture: Optional[str] = False, date_of_birth: Optional[Union[str, datetime, date]] = False, location_name: Optional[str] = False, carrier: Optional[str] = False, device_type: Optional[str] = False, login_type: Optional[str] = False, facebook_url: Optional[Union[str, int]] = False, google_url: Optional[Union[str, int]] = False*) → Tuple[bool, *Profile*]

Update your profile details.

- The default of the parameters is `False`. if you leave it `False`, the parameter will not be updated.

Examples

```
>>> is_success, new_profile = me.update_profile_details(first_name='Chandler', last_
↪name='Bing', date_of_birth='1968-04-08')
>>> new_details = {'location_name': 'New York', 'gender': 'M'}
>>> me.update_profile_details(**new_details)  # dict unpacking
(True, Profile(name='Chandler Bing', date_of_birth=datetime.date(1968, 4, 8),␣
↪location_name='New York', gender='M', ...))
```

Parameters

- **first_name** (str | None) – First name.
- **last_name** (str | None) – Last name.
- **email** (str | None) – For example: user@domian.com.
- **gender** (str | None) – M for male, F for female.
- **profile_picture** (str | None) – Direct image url or local image path. for example: https://example.com/image.png, /home/david/Downloads/my_profile.jpg.
- **slogan** (str | None) – Your bio.
- **date_of_birth** (str | date | datetime | None) – date/datetime obj or string with YYYY-MM-DD format. for example: 1994-09-22.
- **location_name** (str | None) – Your location, can be anything.
- **login_type** (str | None) – email or apple.
- **device_type** (str | None) – android or ios.
- **carrier** – The carrier of your phone. like HOT-mobile, AT&T etc.
- **facebook_url** (str | int | None) – facebook id, for example: 24898745174639.
- **google_url** (str | int | None) – google id, for example: 24898745174639.

Returns
Tuple of: Is update success, new *Profile* object.

Return type
Tuple[bool, *Profile*]

Raises

- **ValueError** – If one of the parameters is not valid.
- ***BlockedAccount*** – If your account is blocked for updating profile details.

Social

Me.**get_socials**(*uuid: Union[str, Profile, User, Contact] = None*) → *Social*

Get connected social networks to Me account.

```
>>> me.get_socials() # get your socials
>>> me.get_socials(uuid='f7930d0f-c8ba-425b-8478-013968f30466') # get socials of␣
↪other user
```

Parameters
uuid (str | *Profile* | *User* | *Contact*) – uuid of the user or *Profile*, *User*, or *Contact* objects. *Default:* Your uuid.

Returns
Social object with social media accounts.

Return type
Social

Raises
ContactHasNoUser – If the contact has no user.

Me.**add_social**(*twitter_token: str = None*, *spotify_token: str = None*, *instagram_token: str = None*, *facebook_token: str = None*, *tiktok_token: str = None*, *pinterest_url: str = None*, *linkedin_url: str = None*) → bool

Connect social network to your me account.

- As of this moment, it is not possible to connect to networks that require a token (all except LinkedIn and Pinterest) because the login url is generted every time. This operation is only possible through the official app.
- If you have at least 2 socials, you get `is_verified=True` in your profile (Blue check).
- The connected socials will be shown in your profile unless you hide them with *switch_social_status()*.

```
>>> me.add_social(pinterest_url='https://www.pinterest.com/username/')
>>> me.add_social(linkedin_url='https://www.linkedin.com/in/username')
>>> me.get_my_profile().is_verified
True
```

Parameters
- **twitter_token** (str) – Twitter Token. Default = None.
- **spotify_token** (str) – Default = None.
- **instagram_token** (str) – Default = None.
- **facebook_token** (str) – Default = None.
- **tiktok_token** (str) – Default = None.
- **pinterest_url** (str) – Profile url - `https://www.pinterest.com/username/`. Default = None.
- **linkedin_url** (str) – Profile url - `https://www.linkedin.com/in/username`. Default = None.

Returns
Is connected successfully.

Return type
bool

Raises
- **TypeError** – If you don't provide any social.
- **ValueError** – If you provide an invalid url.

Me.**remove_social**(*twitter: bool = False*, *spotify: bool = False*, *instagram: bool = False*, *facebook: bool = False*, *pinterest: bool = False*, *linkedin: bool = False*, *tiktok: bool = False*) → bool

Remove social networks from your profile.

- You can also hide social instead of deleting it: *switch_social_status()*.

```
>>> me.remove_social(pinterest=True, linkedin=True)
```

Parameters

- **twitter** (bool) – To remove Twitter. Default: False.
- **spotify** (bool) – To remove Spotify. Default: False.
- **instagram** (bool) – To remove Instagram. Default: False.
- **facebook** (bool) – To remove Facebook. Default: False.
- **pinterest** (bool) – To remove Pinterest. Default: False.
- **linkedin** (bool) – To remove Linkedin. Default: False.
- **tiktok** (bool) – To remove Tiktok. Default: False.

Returns
Is removal success.

Return type
bool

Raises
TypeError – If you don't provide any social.

Me.**switch_social_status**(*twitter: bool = None*, *spotify: bool = None*, *instagram: bool = None*, *facebook: bool = None*, *tiktok: bool = None*, *pinterest: bool = None*, *linkedin: bool = None*) → bool

Switch social network status: Show (True) or Hide (False).

- You can also delete social instead of hiding it: *remove_social()*.

```
>>> me.switch_social_status(pinterest=False, linkedin=False)
>>> me.get_socials().linkedin.is_hidden
True
```

Parameters

- **twitter** (bool) – Switch Twitter status. Default: None.
- **spotify** (bool) – Switch Spotify status Default: None.
- **instagram** (bool) – Switch Instagram status Default: None.
- **facebook** (bool) – Switch Facebook status Default: None.
- **tiktok** (bool) – Switch TikTok status Default: None.
- **pinterest** (bool) – Switch Pinterest status Default: None.
- **linkedin** (bool) – Switch Linkedin status Default: None.

Returns
is switch success (you get True even if social active or was un/hidden before).

Return type
bool

Raises
TypeError – If you don't provide any social.

Me.**friendship**(*phone_number: Union[int, str]*) → *Friendship*

Get friendship information between you and another number. like count mutual friends, total calls duration, how do you name each other, calls count, your watches, comments, and more.

```
>>> me.friendship(972544444444)
Friendship(calls_duaration=0, he_named='Chandler Bing', i_named='Monica Geller'...)
```

Parameters
phone_number (int | str) – International phone number format.

Returns
Friendship object.

Return type
Friendship

Me.**who_deleted**(*incognito: bool = False*, *sorted_by: Optional[str] = 'created_at'*) → List[*Deleter*]

Get list of users who deleted you from their contacts.

The who_deleted_enabled setting must be True in your settings account in order to see who deleted you. See *change_settings()*. You can use incognito=True to automatically enable and disable before and after (Required two more API calls).

```
>>> me.who_deleted(incognito=True)
[Deleter(created_at=datetime.datetime(2020, 1, 1, 0, 0), user=User(uuid='...', name=
↪'...', ...))]
```

Parameters

- **incognito** (bool) – If True, this automatically enables and disables who_deleted_enabled. *Default:* False.
- **sorted_by** (str) – Sort by created_at or None. *Default:* created_at.

Returns
List of Deleter objects sorted by their creation time.

Return type
List[*Deleter*]

Raises
ValueError – If sorted_by is not created_at or None.

Me.**who_watched**(*incognito: bool = False*, *sorted_by: str = 'count'*) → List[*Watcher*]

Get list of users who watched your profile.

The who_watched_enabled setting must be True in your settings account in order to see who watched your profile. See *change_settings()*. You can use incognito=True to automatically enable and disable before and after (Required two more API calls).

```
>>> me.who_watched(incognito=True)
[Watcher(count=1, last_view=datetime.datetime(2020, 1, 1, 0, 0), user=User(uuid='...
↪', name='...', ...))]
```

Parameters

- **incognito** (bool) – If `True`, this automatically enables and disables `who_watched_enabled`. *Default:* `False`.
- **sorted_by** (str) – Sort by `count` or `last_view`. *Default:* `count`.

Returns
List of Watcher objects sorted by watch count (By default) or by last view.

Return type
List[*Watcher*]

Raises
ValueError – If `sorted_by` is not `count` or `last_view`.

`Me.`**suggest_turn_on_mutual**(*uuid: Union[str, Profile, User, Contact]*) → bool

Ask another user to turn on mutual contacts on his profile.

```
>>> me.suggest_turn_on_mutual('d4c7b2c0-5b5a-4b4b-9c1c-8c7b6a5b4c3d')
```

Parameters
uuid (str | *Profile* | *User* | *Contact*) – uuid, *Profile*, *User*, or *Contact* of the user.

Returns
Is request success.

Return type
bool

Raises
ContactHasNoUser – If you provide *Contact* without user.

`Me.`**report_spam**(*country_code: str*, *spam_name: str*, *phone_number: Union[str, int]*) → bool

Report spam on another phone number.

- You get notify when your report is approved. See *get_notifications()*.

```
>>> me.report_spam('IL', 'Spammer', 972544444444)
```

Parameters

- **country_code** (str) – Two letters code, `IL`, `IT`, `US` etc. // Country codes.
- **spam_name** (str) – The spam name that you want to give to the spammer.
- **phone_number** (int | str) – spammer phone number in international format.

Returns
Is report success

Return type
bool

Me.**is_spammer**(*phone_number: Union[int, str]*) → int

Check on phone number if reported as spam.

```
>>> me.is_spammer(989123456789)
```

Parameters
phone_number (int | str) – International phone number format.

Returns
count of spam reports. 0 if None.

Return type
int

Me.**get_age**(*uuid: Union[str, Profile, User, Contact] = None*) → int

Get user age. calculate from date_of_birth, provided by *get_profile()*.

```
>>> me.get_age()
18
```

Parameters
uuid (str | *Profile* | *User* | *Contact*) – uuid of the user or *Profile*, *User*, or *Contact* objects. *Default:* Your uuid.

Returns
User age if date of birth exists. else - 0.

Return type
int

Me.**numbers_count**() → int

Get total count of numbers on Me.

```
>>> me.numbers_count()
6421368062
```

Returns
total count.

Return type
int

Group names

Me.**get_groups**(*sorted_by: str = 'count'*) → List[*Group*]

Get groups of names and see how people named you.

- For more information about Group

```
>>> me.get_groups(sorted_by='count')
[Group(name='Delivery Service', count=2, contacts=[User(name='David'...), User(name=
↪'John'...)]), ...]
```

Parameters
sorted_by (str) – Sort by `count` or `last_contact_at`. *Default:* `count`.

Returns
List of *Group* objects.

Return type
List[*Group*]

Raises
ValueError – If `sorted_by` is not `count` or `last_contact_at`.

`Me.get_deleted_groups()` → List[*Group*]

Get group names that you deleted.

```
>>> me.get_deleted_groups()
[Group(name='Delivery Service', count=2, contacts=[User(name='David'...), User(name=
↪'John'...)]), ...]
```

Returns
List of *Group* objects sorted by their count.

Return type
List[*Group*]

`Me.delete_group`(*contacts_ids: Union[*Group*, int, str, List[Union[int, str]]]*) → bool

Delete group name.

- You can restore deleted group with `restore_name()`.
- You can also ask for rename with *ask_group_rename()*.

```
>>> me.delete_group(contacts_ids=me.get_groups()[0].contact_ids)
```

Parameters
contacts_ids (*Group* | int | str | List[int, str]) – *Group* object, single or list of contact ids from the same group. See *get_groups()*.

Returns
Is delete success.

Return type
bool

`Me.restore_group`(*contacts_ids: Union[int, str, List[Union[int, str]]]*) → bool

Restore deleted group from.

- You can get deleted groups with *get_deleted_groups()*.

```
>>> me.restore_group(contacts_ids=me.get_deleted_groups()[0].contact_ids)
```

Parameters
contacts_ids (*Group* | int | str | List[int, str]) – *Group* object, single or list of contact ids from the same group. See *get_groups()*.

Returns
Is delete success.

Return type
bool

Me.**ask_group_rename**(*contacts_ids: Union[Group, int, str, List[Union[int, str]]]*, *new_name: Optional[str] = None*) → bool

Suggest new name to group of people and ask them to rename you in their contacts book.

```
>>> group = me.get_groups()[0]
>>> group.name
'Delivery Service'
>>> me.ask_group_rename(contacts_ids=group.contact_ids, new_name='Chandler Bing')
```

Parameters

- **contacts_ids** (*Group* | int | str | List[int, str]) – *Group* object, single or list of contact ids from the same group. See *get_groups()*.
- **new_name** (str | None) – Suggested name, Default: Your profile name from *get_profile()*.

Returns
Is asking success.

Return type
bool

Comments

Me.**get_comments**(*uuid: Union[str, Profile, User, Contact] = None*) → List[*Comment*]

Get comments in user's profile.

- Call the method with no parameters to get comments in your profile.

```
>>> comments = me.get_comments(uuid='f7930d0f-c8ba-425b-8478-013968f30466')
[Comment(uuid='...', text='...', ...), ...]
```

Parameters
uuid (str | *User* | *Profile* | *Contact*) – uuid of the user or *Profile*, *User*, or *Contact* objects. *Default:* Your uuid.

Returns
List of *Comment* objects sorted by their like count.

Return type
List[*Comment*]

Raises

- **ValueError** – In the official-auth-method mode you must to provide user uuid if you want to get your comments.
- ***ContactHasNoUser*** – If you provide a contact object without user.

Me.**get_comment**(*comment_id: Union[int, str]*) → *Comment*

Get comment details, comment text, who and how many liked, create time and more.

- This methods return *Comment* object with just `message`, `like_count` and `comment_likes` atrrs.

```
>>> comment = me.get_comment(comment_id=1065393)
```

Parameters
comment_id (`int` | `str`) – Comment id from *get_comments()*.

Returns
Comment object.

Return type
Comment

Me.**publish_comment**(*your_comment: str*, *uuid: Union[str, Profile, User, Contact] = None*, *add_credit: bool = False*) → Optional[*Comment*]

Publish comment in user's profile.

- You can publish comment on your own profile or on someone else's profile.
- When you publish comment on someone else's profile, the user need to approve your comment before it will be visible.
- You can edit your comment by simple calling *publish_comment()* again. The comment status will be changed to `waiting` until the user approves it.
- You can ask the user to enable comments in his profile with *suggest_turn_on_comments()*.
- If the user doesn't enable comments (`comments_enabled`), or if he blocked you from comments (`comments_blocked`), you will get `None`. You can get this info with *get_profile()*.

```
>>> comment = me.publish_comment(your_comment='Hello World!', uuid='f7930d0f-c8ba-
↪425b-8478-013968f30466')
Comment(id=123, status='waiting', message='Hello World!', author=User(uuid='...', ..
↪.), ...)
```

Parameters

- **your_comment** (`str`) – Your comment.
- **uuid** (`str` | *Profile* | *User* | *Contact*) – `uuid` of the user or *Profile*, *User*, or *Contact* objects. *Default:* Your uuid.
- **add_credit** (`bool`) – If `True`, this will add credit to the comment. *Default:* `False`.

Returns
Optional *Comment* object.

Return type
Comment | `None`

Raises
ContactHasNoUser – If you provide a contact object without user.

Me.**approve_comment**(*comment_id: Union[str, int, Comment]*) → bool

Approve comment. the comment will be visible in your profile.

- You can only approve comments that posted on your own profile.
- You can always ignore it with *ignore_comment()*.
- You can approve delete it with *delete_comment()*.
- If the comment already approved, you get `True` anyway.

```
>>> me.approve_comment(comment_id=123)
```

Parameters
comment_id (`str` | `int` | *Comment*) – Comment id from *get_comments()*. or just *Comment* object.

Returns
Is approve success.

Return type
`bool`

Me.**ignore_comment**(*comment_id: Union[str, int, Comment]*) → bool

Ignore comment. the comment will not be visible in your profile.

- you can always approve it with *approve_comment()*.)
- You can only ignore comments that posted on your own profile.

```
>>> me.ignore_comment(comment_id=123)
```

Parameters
comment_id (`int` | `str` | *Comment*) – Comment id from *get_comments()*. or just *Comment* object.

Returns
Is deleting success.

Return type
`bool`

Me.**delete_comment**(*comment_id: Union[str, int, Comment]*) → bool

Delete comment.

- You can only delete comments that posted on your own profile or that posted by you.

```
>>> me.delete_comment(comment_id=123)
```

Parameters
comment_id (`int` | `str` | *Comment*) – Comment id from *get_comments()*. or just *Comment* object.

Returns
Is deleting success.

Return type
bool

Me.**like_comment**(*comment_id: Union[int, str, Comment]*) → bool

Like comment.

- If the comment is already liked, you get True anyway.
- If the comment not approved, you get False.

```
>>> me.like_comment(comment_id=123)
```

Parameters
comment_id (int | str | *Comment*) – Comment id from *get_comments()*. or just *Comment* object.

Returns
Is like success.

Return type
bool

Me.**unlike_comment**(*comment_id: Union[int, str, Comment]*) → bool

Unlike comment.

- If the comment is already unliked, you get True anyway.
- If the comment not approved, you get False.

```
>>> me.unlike_comment(comment_id=123)
```

Parameters
comment_id (int | str | *Comment*) – Comment id from *get_comments()*. or just *Comment* object.

Returns
Is unlike success.

Return type
bool

Me.**block_comments**(*uuid: Union[str, Comment, Profile, User, Contact]*) → bool

Block comments from user.

- If the user is already blocked, you get True anyway.
- There is no apparent way to unblock comments. Use only when you are sure!

```
>>> me.block_comments(uuid='xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
```

Parameters
uuid (str | *User* | *Profile* | *Contact* | *Comment*) – uuid of the user or *Profile*, *User*, *Contact* or *Comment* objects.

Returns
Is block success.

Return type
bool

Raises
ContactHasNoUser – If the contact object has no user.

Me.**suggest_turn_on_comments**(*uuid: Union[str, Profile, User, Contact]*) → bool

Ask another user to turn on comments in his profile.

```
>>> me.suggest_turn_on_comments('d4c7b2c0-5b5a-4b4b-9c1c-8c7b6a5b4c3d')
```

Parameters
uuid (str | *Profile* | *User* | *Contact*) – uuid, *Profile*, *User*, or *Contact* of the commented user.

Returns
Is request success.

Return type
bool

Raises
ContactHasNoUser – If you provide *Contact* without user.

Account

Me.**get_uuid**(*phone_number: Union[int, str] = None*) → Optional[str]

Get user's uuid (To use in *get_profile()*, *get_comments()* and more).

```
>>> me.get_uuid(phone_number=972545416627)
'f7930d0f-c8ba-425b-8478-013968f30466'
```

Parameters
phone_number (str | int | None) – International phone number format. Default: None (Return self uuid).

Returns
String of uuid, or None if no user exists on the provided phone number.

Return type
str | None

Me.**get_saved_contacts**() → List[*User*]

Get all the contacts stored in your contacts (Which has an Me account).

```
>>> saved_contacts = me.get_unsaved_contacts()
>>> for usr in saved_contacts: print(usr.name)
```

Returns
List of saved contacts.

Return type
List[*User*]

Me.**get_unsaved_contacts**() → List[*User*]

Get all the contacts that not stored in your contacts (Which has an Me account).

```
>>> unsaved_contacts = me.get_unsaved_contacts()
>>> for usr in unsaved_contacts: print(usr.name)
```

Returns
List of unsaved contacts.

Return type
List[*User*]

Me.**block_profile**(*phone_number: Union[str, int]*, *block_contact=True*, *me_full_block=True*) → *BlockedNumber*

Block user profile.

```
>>> me.block_profile(phone_number=123456789, block_contact=True, me_full_
↪block=False)
```

Parameters

- **phone_number** (str | int) – User phone number in international format.
- **block_contact** (bool) – To block for calls. *Default:* True.
- **me_full_block** (bool) – To block for social. *Default:* True.

Returns
BlockedNumber object.

Return type
BlockedNumber

Me.**unblock_profile**(*phone_number: int*, *unblock_contact=True*, *me_full_unblock=True*) → bool

Unblock user profile.

```
>>> me.unblock_profile(phone_number=123456789, unblock_contact=True, me_full_
↪unblock=False)
```

Parameters

- **phone_number** (str | int) – User phone number in international format.
- **unblock_contact** (bool) – To unblock for calls. *Default:* True.
- **me_full_unblock** (bool) – To unblock for social. *Default:* True.

Returns
Is successfully unblocked.

Return type
bool

Me.**block_numbers**(*numbers: Union[int, str, List[Union[int, str]]]*) → bool

Block phone numbers.

```
>>> me.block_numbers(numbers=[123456789, 987654321])
```

Parameters
numbers (int | str | List[int | str]) – Single or list of phone numbers in international format.

Returns
Is blocked success.

Return type
bool

Me.**unblock_numbers**(*numbers: Union[int, List[int]]*) → bool

Unblock numbers.

```
>>> me.unblock_numbers(numbers=[123456789, 987654321])
```

Parameters
numbers (int | List[int]) – Single or list of phone numbers in international format. See *get_blocked_numbers()*.

Returns
Is unblocking success.

Return type
bool

Me.**get_blocked_numbers**() → List[*BlockedNumber*]

Get list of your blocked numbers. See *unblock_numbers()*.

```
>>> me.get_blocked_numbers()
[BlockedNumber(phone_number=123456789, block_contact=True, me_full_block=True]
```

Returns
List of blocked_number.BlockedNumber objects.

Return type
List[*BlockedNumber*]

Me.**upload_random_data**(*count: int = 50*, *contacts=False*, *calls=False*, *location=False*) → bool

Upload random data to your account.

```
>>> me.upload_random_data(count=50, contacts=True, calls=True, location=True)
```

Parameters

- **count** (int) – Count of random data to upload. Default: 50.
- **contacts** (bool) – To upload random contacts data. Default: False.
- **calls** (bool) – To upload random calls data. Default: False.
- **location** (bool) – To upload random location data. Default: False.

Returns
Is uploading success.

Return type
bool

Me.**suspend_account**(*yes_im_sure: bool = False*) → bool

Suspend your account until your next login.

```
>>> me.suspend_account(yes_im_sure=True)
```

Parameters
yes_im_sure (bool) – True to suspend your account and ignore prompt. *Default:* False.

Returns
Is suspended.

Return type
bool

Me.**delete_account**(*yes_im_sure: bool = False*) → bool

Delete your account and it's data (!!!)

```
>>> me.delete_account(yes_im_sure=True)
True
```

Parameters
yes_im_sure (bool) – True to delete your account and ignore prompt. *Default:* False.

Returns
Is deleted.

Return type
bool

Me.**add_contacts**(*contacts: List[Dict[str, Optional[Union[str, int]]]]*) → dict

Upload new contacts to your Me account. See *upload_random_data()*.

```
>>> contacts = [{'country_code': 'XX', 'date_of_birth': None, 'name': 'Chandler',
↪'phone_number': 512145887}]
>>> me.add_contacts(contacts=contacts)
```

Parameters
contacts (List[dict])) – List of dicts with contacts data.

Returns
Dict with upload results.

Return type
dict

Example of list of contacts to add:

```
[
    {
        "country_code": "XX",
        "date_of_birth": None,
        "name": "Chandler",
        "phone_number": 512145887,
    }
]
```

Me.**remove_contacts**(*contacts: List[Dict[str, Optional[Union[str, int]]]]*) → dict

Remove contacts from your Me account.

```
>>> contacts = [{'country_code': 'XX', 'date_of_birth': None, 'name': 'Chandler',
↪'phone_number': 512145887}]
>>> me.remove_contacts(contacts=contacts)
```

Parameters
contacts (List[dict]) – List of dicts with contacts data.

Returns
Dict with upload results.

Return type
dict

Example of list of contacts to remove:

```
[
    {
        "country_code": "XX",
        "date_of_birth": None,
        "name": "Chandler",
        "phone_number": 512145887,
    }
]
```

Me.**add_calls_to_log**(*calls: List[Dict[str, Optional[Union[str, int]]]]*) → List[Call]

Add call to your calls log. See *upload_random_data()*.

```
>>> calls = [{'called_at': '2021-07-29T11:27:50Z', 'duration': 28, 'name':
↪'043437535', 'phone_number': 43437535, 'tag': None, 'type': 'missed'}]
>>> me.add_calls_to_log(calls=calls)
```

Parameters
calls (List[dict]) – List of dicts with calls data.

Returns
dict with upload result.

Return type
dict

Example of list of calls to add:

```
[
    {
        "called_at": "2021-07-29T11:27:50Z",
        "duration": 28,
        "name": "043437535",
        "phone_number": 43437535,
        "tag": None,
        "type": "missed",
    }
]
```

Me.**remove_calls_from_log**(*calls: List[Dict[str, Optional[Union[str, int]]]]*) → List[Call]

Remove calls from your calls log.

```
>>> calls = [{'called_at': '2021-07-29T11:27:50Z', 'duration': 28, 'name':
↪'043437535', 'phone_number': 43437535, 'tag': None, 'type': 'missed'}]
>>> me.remove_calls_from_log(calls=calls)
```

Parameters
calls (List[dict]) – List of dicts with calls data.

Returns
dict with upload result.

Return type
dict

Example of list of calls to remove:

```
[
    {
        "called_at": "2021-07-29T11:27:50Z",
        "duration": 28,
        "name": "043437535",
        "phone_number": 43437535,
        "tag": None,
        "type": "missed",
    }
]
```

Location

Me.**update_location**(*latitude: float*, *longitude: float*) → bool

Update your location. See *upload_random_data()*.

```
>>> me.update_location(35.6892, 51.3890)
```

Parameters

- **latitude** (float) – location latitude coordinates.
- **longitude** (float) – location longitude coordinates.

Returns
Is location update success.

Return type
bool

Raises
ValueError – If latitude or longitude is not a float.

Me.**suggest_turn_on_location**(*uuid: Union[str, Profile, User, Contact]*) → bool

Ask another user to share his location with you.

```
>>> me.suggest_turn_on_location('d4c7b2c0-5b5a-4b4b-9c1c-8c7b6a5b4c3d')
```

Parameters
uuid (str | *Profile* | *User* | *Contact*) – uuid, *Profile*, *User*, or *Contact* of the user.

Returns
Is request success.

Return type
bool

Raises
ContactHasNoUser – If you provide *Contact* without user.

Me.**share_location**(*uuid: Union[str, Profile, User, Contact]*) → bool

Share your location with another user.

```
>>> me.share_location('d4c7b2c0-5b5a-4b4b-9c1c-8c7b6a5b4c3d')
```

Parameters
uuid (str | *Profile* | *User* | *Contact*) – uuid of the user or *Profile*, *User*, or *Contact* objects.

Returns
Is sharing success.

Return type
bool

Me.**stop_sharing_location**(*uuids: Union[str, Profile, User, Contact, List[Union[str, Profile, User, Contact]]]*) → bool

Stop sharing your *update_location()* with users.

```
>>> me.stop_sharing_location('d4c7b2c0-5b5a-4b4b-9c1c-8c7b6a5b4c3d')
```

Parameters
uuids (str | *Profile* | *User* | *Contact* | List[str | *Profile* | *User* | *Contact*]) – uuid/s of the user/s that you want to stop sharing your location with.

Returns
is stopping success.

Return type
bool

Me.**stop_shared_location**(*uuids: Union[str, Profile, User, Contact, List[Union[str, Profile, User, Contact]]]*) → bool

Stop locations that shared with you.

```
>>> me.stop_shared_location('d4c7b2c0-5b5a-4b4b-9c1c-8c7b6a5b4c3d')
```

Parameters
uuids (str | *Profile* | *User* | *Contact* | List[str | *Profile* | *User* | *Contact*]) – uuid/s of the user/s that you want to stop sharing your location with.

Returns
is stopping success.

Return type
bool

Me.**locations_shared_by_me**() → List[*User*]

Get list of users that you shared your location with them.

```
>>> me.locations_shared_by_me()
[User(name='John Doe', uuid='d4c7b2c0-5b5a-4b4b-9c1c-8c7b6a5b4c3d', ...)]
```

Returns
List of *User* objects.

Return type
List[*User*]

Me.**locations_shared_with_me**() → List[*User*]

Get users who have shared their location with you.

```
>>> me.locations_shared_with_me()
[User(name='John Doe', uuid='d4c7b2c0-5b5a-4b4b-9c1c-8c7b6a5b4c3d', ...)]
```

Returns
List of *User* objects (with `distance` attribute).

Return type
List[*User*]

Notifications

Me.**unread_notifications_count**() → int

Get count of unread notifications.

```
>>> me.unread_notifications_count()
25
```

Returns
count of unread notifications.

Return type
int

Me.**get_notifications**(*page: int = 1*, *limit: int = 20*, *names_filter: bool = False*, *system_filter: bool = False*, *comments_filter: bool = False*, *who_watch_filter: bool = False*, *who_deleted_filter: bool = False*, *birthday_filter: bool = False*, *location_filter: bool = False*) → List[*Notification*]

Get app notifications: new names, birthdays, comments, watches, deletes, location shares and system notifications.

```
>>> me.get_notifications(limit=300, names_filter=True, system_filter=True)
[Notification(id=109, category='JOINED_ME', is_read=False, created_at=datetime.
↪datetime(2020, 1, 1), ...),]
```

Parameters

- **page** (int) – *get_notifications()*.``count`` / page_size. *Default:* 1.
- **limit** (int) – Limit of notifications in each page. *Default:* 20.
- **names_filter** (bool) – New names, deletes, joined, renames, rename requests. *Default:* False.
- **system_filter** (bool) – System notifications: spam reports, your name requests, suggestions to turn on mutual contacts. *Default:* False.
- **comments_filter** (bool) – Comments notifications: new comments, published comments and suggestions to turn on comments (See *get_comments()*). *Default:* False.
- **who_watch_filter** (bool) – Who watched your profile (See *who_watched()*). *Default:* False.
- **who_deleted_filter** (bool) – Who deleted you from his contacts (See *who_deleted()*). *Default:* False.
- **birthday_filter** (bool) – Contacts birthdays. *Default:* False.
- **location_filter** (bool) – Shared locations: suggestions to turn on location, locations that shared with you. *Default:* False.

Returns
List of *Notification* objects.

Return type
List[*Notification*]

Me.**read_notification**(*notification_id: Union[int, str,* Notification*]*) → bool

Mark notification as read.

```
>>> me.read_notification(109)
True
```

Parameters
notification_id (int | str | *Notification*) – Notification id from *get_notifications()* or *Notification* object.

Returns
Is read success.

Return type
bool

Settings

Me.**get_settings**() → *Settings*

Get current settings.

```
>>> # Take control of your privacy:
>>> my_settings = me.get_settings()
>>> my_settings.who_watched_enabled = False
>>> my_settings.who_deleted_enabled = False
>>> my_settings.mutual_contacts_available = False
```

(continues on next page)

(continued from previous page)

```
>>> my_settings.comments_enabled = False
>>> my_settings.location_enabled = False
```

Returns
Settings object.

Return type
Settings

Me.**change_settings**(*mutual_contacts_available: bool = None*, *who_watched_enabled: bool = None*, *who_deleted_enabled: bool = None*, *comments_enabled: bool = None*, *location_enabled: bool = None*, *language: str = None*, *who_deleted_notification_enabled: bool = None*, *who_watched_notification_enabled: bool = None*, *distance_notification_enabled: bool = None*, *system_notification_enabled: bool = None*, *birthday_notification_enabled: bool = None*, *comments_notification_enabled: bool = None*, *names_notification_enabled: bool = None*, *notifications_enabled: bool = None*) → Tuple[bool, *Settings*]

Change social, app and notification settings.

```
>>> me.change_settings(language='en', mutual_contacts_available=False)
(True, Settings(language='en', mutual_contacts_available=False, ...))
```

Parameters

- **mutual_contacts_available** (bool) – Show common contacts between users. *Default:* None.
- **who_watched_enabled** (bool) – Users will be notified that you have viewed their profile. *Default:* None. - They will only be able to get information about you if they are premium users (is_premium = True in *get_profile()*) or, by using meapi ;) - This setting must be True if you want to use *who_watched()* method.
- **who_deleted_enabled** (bool) – Users will be notified that you have deleted them from your contact book. *Default:* None. - They will only be able to get information about you if they are premium users (is_premium = True in *get_profile()*) or, by using meapi ;) - This setting must be True if you want to use *who_deleted()* method.
- **comments_enabled** (bool) – Allow users to publish comment (*publish_comment()*) in your profile. *Default:* None. - Comments will not be posted until you approve them with *approve_comment()*.
- **location_enabled** (bool) – Allow shared locations. *Default:* None.
- **language** (str) – lang code: iw, en, etc. (For notifications text). *Default:* None.
- **who_deleted_notification_enabled** (bool) – *Default:* None.
- **who_watched_notification_enabled** (bool) – *Default:* None.
- **distance_notification_enabled** (bool) – *Default:* None.
- **system_notification_enabled** (bool) – *Default:* None.
- **birthday_notification_enabled** (bool) – *Default:* None.
- **comments_notification_enabled** (bool) – *Default:* None.
- **names_notification_enabled** (bool) – *Default:* None.
- **notifications_enabled** (bool) – *Default:* None.

Raises
TypeError – If you don't change any setting.

Returns
Tuple: Is success, *Settings* object.

Return type
Tuple[bool, *Settings*]

Advanced

Me.**login**(*activation_code: Optional[str] = None*, *new_account_details: Optional[NewAccountDetails] = None*, *interactive_mode: bool = False*) → bool

Login to MeApi.

- If you initialized the Me class directly, this method will be called automatically. No need to call it.
- If activation_code is not provided and interactive_mode is True the method will prompt for activation code.
- If the account is new and new_account_details is not provided and interactive_mode is True the method will prompt for new account details.

Parameters

- **activation_code** (str | None) – You can pass the activation code if you want to skip the prompt. *Default:* None.
- **new_account_details** (NewAccountDetails | None) – You can pass the new account details if you want to skip the prompt in case of new account. Default: None.
- **interactive_mode** (bool) – If True the method will prompt for activation code (and new account details in case of new account). *Default:* False.

Raises

- *NeedActivationCode* – If activation_code is not provided and interactive_mode is False.
- *IncorrectActivationCode* – If the activation code is incorrect and interactive_mode is False.
- *ActivationCodeExpired* – If the activation code is expired and interactive_mode is False.
- *MaxValidateReached* – If activation_code is incorrect for a few times.
- *NewAccountException* – If the account is new and new_account_details is not provided and interactive_mode is False.
- *BrokenCredentialsManager* – If the credentials manager is broken (if authentication succeeded but the credentials manager failed to save the credentials).

Returns
Is login succeeded.

Return type
bool

Me.**logout**() → bool

Logout from Me account (And delete credentials, depends on the credentials manager implementation).

- If you want to login again, you need to call *login()* or create a new instance of *Me*.
- If you try to use any method that requires authentication, you will get a *NotLoggedIn* exception.

Returns
Is success.

Type
bool

Me.**make_request**(*method: Union[str,* RequestType*]*, *endpoint: str*, *body: Dict[str, Any] = None*, *headers: Dict[str, Union[str, int]] = None*, *files: Dict[str, bytes] = None*, *max_retries: int = 3*) → Union[Dict[str, Any], List[Dict[str, Any]]]

Make a raw request to Me API.

```
>>> me.make_request('GET', '/main/users/profile/me/')
>>> me.make_request('PATCH', '/main/users/profile/', body={'name': 'Chandler Bing'})
```

Parameters

- **method** (str | RequestType) – Request method. Can be GET, POST, PUT, DELETE, PATCH, HEAD or OPTIONS.
- **endpoint** (str) – API endpoint. e.g. /main/users/profile/me/.
- **body** (dict) – The body of the request. Default: None.
- **headers** (dict) – Use different headers instead of the default ones. Default: None.
- **files** (dict) – Files to send with the request. Default: None.
- **max_retries** (int) – Maximum number of retries in case of failure. Default: 3.

Raises

- *MeApiException* – If HTTP status is higher than 400.
- ValueError – If the response received does not contain a valid JSON or the request type is not valid.
- ConnectionError – If the request failed.

Returns
API response as dict or list.

Return type
dict | list

7.2.2 Raw

The next page contains raw functions that accept an instance of `Me` in the first parameter, make an API call and return raw data. meapi maps the data to objects, but these functions can be accessed manually:

```
from meapi.api.raw.account import phone_search_raw, get_profile_raw
from meapi import Me

me = Me(972123456789) # initialize the client

search_res = phone_search_raw(me, 972987654321)
profile_res = get_profile_raw(me, 'fdsfs-fdsfs-fdsfs-fdsfs')
```

Auth

meapi.api.raw.auth.**activate_account_raw**(*client:* Me, *phone_number:* *int*, *activation_code:* *str*) → Dict[str, str]

Activate your account with the activation code.

Parameters

- **client** (*Me*) – *Me* client object.
- **phone_number** (int) – International phone number format.
- **activation_code** (str) – Activation code.

Return type
dict

Example:

```
{
    "access": "eyxxxxKV1xxxxxx1NiJ9.xxx.xxx",
    "pwd_token": "xxxx-xxxx-xxxxx-xxxx-xxxxxxxxxxd2",
    "refresh": "xxxx.xxxxxx.xxxx-xxxxx-xxxx"
}
```

meapi.api.raw.auth.**ask_for_call_raw**(*client:* Me, *phone_number:* *str*, *session_token:* *str*) → dict

Ask Me to call you with the activation code.

Parameters

- **client** (*Me*) – *Me* client object.
- **phone_number** (int) – International phone number format.
- **session_token** (str) – Session token.

Return type
dict

Example:

```
{
    "activation_type": "failb",
    "error_message": "",
    "provider_name": "TwilioCall",
```

(continues on next page)

(continued from previous page)

```
    "success": True
}
```

meapi.api.raw.auth.**ask_for_sms_raw**(*client:* Me, *phone_number: str*, *session_token: str*) → Dict[str, Union[str, bool]]

Ask me to send you the activation code in SMS.

Parameters

- **client** (*Me*) – *Me* client object.
- **phone_number** (int) – International phone number format.
- **session_token** (str) – Session token.

Return type
dict

Example:

```
{
    "activation_type": "sms",
    "error_message": "",
    "provider_name": "IsraelSms",
    "success": True
}
```

meapi.api.raw.auth.**generate_new_access_token_raw**(*client:* Me, *phone_number: str*, *pwd_token: str*) → dict

Get new access_token.

Parameters

- **client** (*Me*) – *Me* client object.
- **phone_number** (int) – International phone number format.
- **pwd_token** (str) – The pwd_token from the first activation (activate_account_raw).

Return type
dict

Example:

```
{
    "access": "eyxxxxKV1xxxxxx1NiJ9.xxx.xxx",
    "refresh": "xxxx.xxxxxx.xxxx-xxxxx-xxxx"
}
```

Account

meapi.api.raw.account.**add_calls_raw**(*client:* Me, *calls: List[dict]*) → dict

Add call to your calls log. See `upload_random_data()`.

Parameters

- **client** (*Me*) – *Me* client object.
- **calls** (List[dict]) – List of dicts with calls data.

Returns
dict with upload result.

Return type
dict

Example of list of calls to add:

```
[
    {
        "called_at": "2021-07-29T11:27:50Z",
        "duration": 28,
        "name": "043437535",
        "phone_number": 43437535,
        "tag": None,
        "type": "missed",
    }
]
```

meapi.api.raw.account.**add_contacts_raw**(*client:* Me, *contacts: List[dict]*) → dict

Upload new contacts to your Me account.

Parameters

- **client** (*Me*) – *Me* client object.
- **contacts** (List[dict]) – List of contacts to add.

Return type
dict

Example of list of contacts to add:

```
[
    {
        "country_code": "XX",
        "date_of_birth": None,
        "name": "Chandler",
        "phone_number": 512145887,
    }
]
```

Example of results:

```
{
    'total': 1,
    'added': 1,
    'updated': 0,
```

(continues on next page)

(continued from previous page)

```
    'removed': 0,
    'failed': 0,
    'same': 0,
    'result':
        [{
            'phone_number': 512145887,
            'name': 'Chandler',
            'email': None,
            'referenced_user': None,
            'created_at': '2022-06-25T22:28:41:955339Z',
            'modified_at': '2022-06-25T22:28:41Z',
            'country_code': 'XX',
            'date_of_birth': None
        }],
    'failed_contacts': []
}
```

meapi.api.raw.account.**block_numbers_raw**(*client:* Me, *numbers: List[int]*) → dict

Block numbers.

Parameters

- **client** (*Me*) – *Me* client object.
- **numbers** (List[int]) – Single or list of phone numbers in international format.

Returns

list of dicts with the blocked numbers.

Return type

List[dict]

Example:

```
[
    {
        "block_contact": True,
        "me_full_block": False,
        "phone_number": 1234567890
    }
]
```

meapi.api.raw.account.**block_profile_raw**(*client:* Me, *phone_number: int*, *block_contact: bool*, *me_full_block: bool*) → dict

Block user profile.

Parameters

- **client** (*Me*) – *Me* client object.
- **phone_number** (int | str) – User phone number in international format.
- **block_contact** (bool) – To block for calls.
- **me_full_block** (bool) – To block for social.

Returns

Dict of results.

Return type
dict

Example of results:

```
{
    'success': True,
    'message': 'Successfully block  updated'
}
```

meapi.api.raw.account.**delete_account_raw**(*client:* Me) → dict

Delete your account.

Parameters
client (*Me*) – *Me* client object.

Return type
dict

Example:

```
{}
```

meapi.api.raw.account.**get_blocked_numbers_raw**(*client:* Me) → List[dict]

Get your blocked numbers.

Parameters
client (*Me*) – *Me* client object.

Returns
list of dicts.

Return type
List[dict]

Example:

```
[
    {
        "block_contact": True,
        "me_full_block": False,
        "phone_number": 1234567890
    }
]
```

meapi.api.raw.account.**get_my_profile_raw**(*client:* Me) → dict

Get your profile.

Parameters
client (*Me*) – *Me* client object.

Return type
dict

Example:

```
{
    'first_name': 'Ross geller',
    'last_name': '',
```

(continues on next page)

(continued from previous page)

```
    'facebook_url': '123456789',
    'google_url': None,
    'email': 'ross@friends.tv',
    'profile_picture': 'https://d18zaexen4dp1s.cloudfront.net/dXXXXXXXXXXXXX26b.jpg
↪',
    'date_of_birth': '9999-12-12',
    'gender': None,
    'location_latitude': -37.57539,
    'location_longitude': 31.30874,
    'location_name': 'Argentina',
    'phone_number': 387648734435,
    'is_premium': False,
    'is_verified': False,
    'uuid': '3XXXb-3f7e-XXXX-XXXXX-XXXXX',
    'slogan': 'Pivot!',
    'device_type': 'ios',
    'carrier': 'BlaMobile',
    'country_code': 'AR',
    'phone_prefix': '387',
    'gdpr_consent': True,
    'login_type': 'apple',
    'verify_subscription': True
}
```

meapi.api.raw.account.**get_profile_raw**(*client:* Me, *uuid:* *str* = *None*) → dict

Get other users profile.

Parameters

- **client** (*Me*) – *Me* client object.
- **uuid** (str) – uuid of the Me user..

Returns

Dict with profile details

Return type

dict

Example:

```
{
    "comments_blocked": False,
    "is_he_blocked_me": False,
    "is_permanent": False,
    "is_shared_location": False,
    "last_comment": None,
    "mutual_contacts_available": True,
    "mutual_contacts": [
        {
            "phone_number": 1234567890,
            "name": "Ross geller",
            "referenced_user": {
                "email": "rossgeller@friends.com",
                "profile_picture": "https://d18zaexen4dp1s.cloudfront.net/
```

(continues on next page)

(continued from previous page)

```
↪59XXXXXXXXXXXXXXXX67.jpg",
                "first_name": "Ross",
                "last_name": "",
                "gender": 'M',
                "uuid": "XXXX-XXX-XXX-83c1-XXXX",
                "is_verified": True,
                "phone_number": 3432434546546,
                "slogan": "Pivot!",
                "is_premium": False,
                "verify_subscription": True,
            },
            "date_of_birth": '1980-03-13',
        }
    ],
    "profile": {
        "carrier": "XXX mobile",
        "comments_enabled": False,
        "country_code": "XX",
        "date_of_birth": '2222-05-20',
        "device_type": "android",
        "distance": None,
        "email": "user@domain.com",
        "facebook_url": "133268763438473",
        "first_name": "Chandler",
        "gdpr_consent": True,
        "gender": 'M',
        "google_url": None,
        "is_premium": False,
        "is_verified": True,
        "last_name": "Bing",
        "location_enabled": False,
        "location_name": "XXXX",
        "login_type": "email",
        "me_in_contacts": True,
        "phone_number": 123456789012,
        "phone_prefix": "123",
        "profile_picture": "https://d18zaexen4dp1s.cloudfront.net/
↪5XXX712a0676XXXXXXXfa67.jpg",
        "slogan": "I will always be there for you",
        "user_type": "BLUE",
        "uuid": "XXXXXXXXXXXXXXXXXX3c1-6932bc9eb597",
        "verify_subscription": True,
        "who_deleted_enabled": True,
        "who_watched_enabled": True,
    },
    "share_location": False,
    "social": {
        "facebook": {
            "posts": [],
            "profile_id": "https://www.facebook.com/app_scoped_user_id/XXXXXXXXXX/
↪",
            "is_active": True,
```

(continues on next page)

(continued from previous page)


```
            "is_hidden": True,
        },
        "fakebook": {
            "is_active": False,
            "is_hidden": True,
            "posts": [],
            "profile_id": None,
        },
        "instagram": {
            "posts": [
                {
                    "posted_at": "2021-12-23T22:21:06Z",
                    "photo": "https://d18zaexen4dp1s.cloudfront.net/XXXXXXXXXXXXX.
↪jpg",
                    "text_first": None,
                    "text_second": "IMAGE",
                    "author": "username",
                    "redirect_id": "CXXXXIz-0",
                    "owner": "username",
                }
            ],
            "profile_id": "username",
            "is_active": True,
            "is_hidden": False,
        },
        "linkedin": {
            "is_active": True,
            "is_hidden": False,
            "posts": [],
            "profile_id": "https://www.linkedin.com/in/username",
        },
        "pinterest": {
            "posts": [],
            "profile_id": "https://pin.it/XXXXXXXX",
            "is_active": True,
            "is_hidden": False,
        },
        "spotify": {
            "is_active": True,
            "is_hidden": False,
            "posts": [
                {
                    "author": "Chandler bing",
                    "owner": "4xgXXXXXXXt0pv",
                    "photo": "https://d18zaexen4dp1s.cloudfront.net/
↪9bcXXXfa7dXXXXXXXac.jpg",
                    "posted_at": None,
                    "redirect_id": "4KgES5cs3SnMhuAXuBREW2",
                    "text_first": "My friends playlist songs",
                    "text_second": "157",
                },
                {
```


(continues on next page)

(continued from previous page)


```
                    "author": "Chandler Bing",
                    "owner": "4xgoXcoriuXXXXpt0pv",
                    "photo": "https://d18zaexen4dp1s.cloudfront.net/
↪55d3XXXXXXXXXXXXXXXXX4.jpg",
                    "posted_at": None,
                    "redirect_id": "3FjSXXXCQPB14Xt",
                    "text_first": "My favorite songs!",
                    "text_second": "272",
                },
            ],
            "profile_id": "4xgot8coriuXXXXXpt0pv",
        },
        "tiktok": {
            "is_active": False,
            "is_hidden": True,
            "posts": [],
            "profile_id": None,
        },
        "twitter": {
            "is_active": True,
            "is_hidden": False,
            "posts": [
                {
                    "author": "username",
                    "owner": "username",
                    "photo": "https://pbs.twimg.com/profile_images/13XXXXX76/
↪AvBXXXX_normal.jpg",
                    "posted_at": "2021-08-24T10:02:45Z",
                    "redirect_id": "https://twitter.com/username/status/1XXXXXX423",
                    "text_first": "My tweet #1 https://t.co/PLXXXX2Tw https://t.co/
↪zXXXXkk",
                    "text_second": None,
                },
                {
                    "author": "username",
                    "owner": "username",
                    "photo": "https://pbs.twimg.com/profile_images/1318XXXX0976/
↪AvBXXXUk_normal.jpg",
                    "posted_at": "2021-08-12T10:09:23Z",
                    "redirect_id": "https://twitter.com/username/status/
↪142XXXXX86624",
                    "text_first": "My second tweet https://t.co/xtqXXXtAC",
                    "text_second": None,
                },
            ],
            "profile_id": "username",
        },
    },
}
```


meapi.api.raw.account.**phone_search_raw**(*client:* Me, *phone_number:* *Union[str, int]*) → dict

Get information on any phone number.

Parameters

- **client** (*Me*) – *Me* client object.
- **phone_number** (int) – International phone number format.

Return type
dict

Example for existed user:

```
{
    "contact": {
        "name": "Chandler bing",
        "picture": None,
        "user": {
            "email": "user@domain.com",
            "profile_picture": "https://d18zaexXXp1s.cloudfront.net/
↪5XXX971XXXXXXXXXfa67.jpg",
            "first_name": "Chandler",
            "last_name": "Bing",
            "gender": 'M',
            "uuid": "XXXXX-XXXX-XXXX-XXXX-XXXX",
            "is_verified": True,
            "phone_number": 7434872457,
            "slogan": "User bio",
            "is_premium": False,
            "verify_subscription": True,
            "id": 42453345,
            "comment_count": 0,
            "location_enabled": False,
            "distance": None,
        },
        "suggested_as_spam": 0,
        "is_permanent": False,
        "is_pending_name_change": False,
        "user_type": "BLUE",
        "phone_number": 7434872457,
        "cached": True,
        "is_my_contact": False,
        "is_shared_location": False,
    }
}
```

Example for non user:

```
{
    "contact": {
        "name": "Chandler bing",
        "picture": None,
        "user": None,
        "suggested_as_spam": 0,
        "is_permanent": False,
        "is_pending_name_change": False,
        "user_type": "GREEN",
        "phone_number": 123456789,
```

(continues on next page)

(continued from previous page)

```
        "cached": False,
        "is_my_contact": False,
        "is_shared_location": False,
    }
}
```

meapi.api.raw.account.**remove_calls_raw**(*client:* Me, *calls: List[dict]*) → dict

Remove call from your calls log

Parameters

- **client** (*Me*) – *Me* client object.
- **calls** (List[dict]) – List of dicts with calls data.

Returns
dict with upload result.

Return type
dict

Example of list of calls to remove:

```
[
    {
        "called_at": "2021-07-29T11:27:50Z",
        "duration": 28,
        "name": "043437535",
        "phone_number": 43437535,
        "tag": None,
        "type": "missed",
    }
]
```

meapi.api.raw.account.**remove_contacts_raw**(*client:* Me, *contacts: List[dict]*) → dict

Remove contacts from your Me account.

Parameters

- **client** (*Me*) – *Me* client object.
- **contacts** (List[dict]) – List of contacts to remove.

Return type
dict

Example of list of contacts to remove:

```
[
    {
        "country_code": "XX",
        "date_of_birth": None,
        "name": "Chandler",
        "phone_number": 512145887,
    }
]
```

Example of results:

```
{
    'total': 1,
    'added': 0,
    'updated': 0,
    'removed': 1,
    'failed': 0,
    'same': 0,
    'result': [],
    'failed_contacts': []
}
```

Parameters
contacts –

meapi.api.raw.account.**suspend_account_raw**(*client:* Me) → dict

Suspend your account.

Parameters
client (*Me*) – *Me* client object.

Return type
dict

Example:

```
{
    'mutual_contacts_available': False,
    'contact_suspended': True,
    'last_backup_at': None,
    'last_restore_at': None,
    'who_watched_enabled': False,
    'comments_enabled': False,
    'language': 'en',
    'notifications_enabled': False,
    'location_enabled': False,
    'names_notification_enabled': False,
    'who_watched_notification_enabled': False,
    'comments_notification_enabled': False,
    'birthday_notification_enabled': False,
    'system_notification_enabled': False,
    'distance_notification_enabled': False,
    'who_deleted_enabled': False,
    'who_deleted_notification_enabled': False
}
```

meapi.api.raw.account.**unblock_numbers_raw**(*client:* Me, *numbers: List[int]*) → dict

Unblock phone numbers.

Parameters

- **client** (*Me*) – *Me* client object.
- **numbers** (List[int]) – Single or list of phone numbers in international format.

Returns
dict with unblock success details.

Return type
dict

Example:

```
{
    'success': True,
    'message': 'Phone numbers successfully unblocked'
}
```

meapi.api.raw.account.**unblock_profile_raw**(*client:* Me, *phone_number:* *int*, *unblock_contact=True*, *me_full_unblock=True*) → dict

Unlock user profile.

Parameters

- **client** (*Me*) – *Me* client object.
- **phone_number** (int) – User phone number in international format.
- **unblock_contact** (bool) – To unblock for calls.
- **me_full_unblock** (bool) – To unblock for social.

Returns
Dict of results.

Return type
dict

Example of results:

```
{
    'success': True,
    'message': 'Successfully block  updated'
}
```

meapi.api.raw.account.**update_fcm_token_raw**(*client:* Me, *fcm_token:* *str = None*) → dict

Update FCM token.

Parameters

- **client** (*Me*) – *Me* client object.
- **fcm_token** (str) – FCM token.

meapi.api.raw.account.**update_profile_details_raw**(*client:* Me, ***kwargs*) → dict

Update your profile details.

Parameters

- **client** (*Me*) – *Me* client object.
- **kwargs** – key value of profile details.

Example:

```
{
    'first_name': 'Joey',
    'last_name': 'Tribbiani',
    'facebook_url': '123456789',
```

(continues on next page)

(continued from previous page)

```
    'google_url': None,
    'email': 'joeyt@friends.tv',
    'profile_picture': 'https://refr.cloudfront.net/dde.jpg',
    'date_of_birth': '1999-01-01',
    'gender': 'M',
    'location_latitude': -12.5465657,
    'location_longitude': 32.454354,
    'location_name': 'XX',
    'phone_number': 95435436545765483,
    'is_premium': False,
    'is_verified': True,
    'uuid': '3850f44b-3f7e-41cf-af6a-27ce13b64d0d',
    'slogan': 'Joey doesnt share food!',
    'device_type': 'ios',
    'carrier': 'MobileBla',
    'country_code': 'US',
    'phone_prefix': '123',
    'gdpr_consent': True,
    'login_type': 'apple',
    'verify_subscription': True
}
```

meapi.api.raw.account.**upload_image_raw**(*client:* Me, *binary_img:* *bytes*) → dict

Upload image to Me servers.

Parameters

- **client** (*Me*) – *Me* client object.
- **binary_img** (bytes) – Binary image.

Returns

dict with upload details.

Return type

dict

Example:

```
{
    'attributes': {
        'image_type': 'jpeg'
    },
    'content_type': 'image/jpeg',
    'created_at': '2022-07-20T20:40:09Z',
    'deletion_msg': None,
    'deletion_status': None,
    'file_hash': '24edcdoisubas7afb94hd8w7a2c38',
    'id': 422374926,
    'is_processed': True,
    'name': '165didw08856_temp.jpg',
    'old_url': None,
    'path': '/meapp-s3-files/ehfe9whufe9ufh9eww.jpg',
    'processed_at': None,
    'size': 0.0,
```

(continues on next page)

(continued from previous page)

```
    'url': 'https://d18zaexen4dp1s.cloudfront.net/24e9a99sdisfiseorew0b5a7a2c38.jpg'
}
```

Social

meapi.api.raw.social.**add_social_token_raw**(*client:* Me, *social_name: str*, *token: str*) → dict

Connect social network (that required token) to your Me account.

- Available social networks: `facebook`, `instagram`, `spotify`, `twitter`, `tiktok`.

Parameters

- **client** (*Me*) – *Me* client object.
- **social_name** (`str`) – Social network name.
- **token** (`str`) – Token from the social network.

Returns
Dict with added success.

Return type
`dict`

Example:

```
{
    "success": True
}
```

meapi.api.raw.social.**add_social_url_raw**(*client:* Me, *social_name: str*, *url: str*) → dict

Connect social network (that required url) to your Me account.

- Available for `linkedin` and `pintrest`.

Parameters

- **client** (*Me*) – *Me* client object.
- **social_name** (`str`) – Social network name.
- **url** (`str`) – Url to your social profile.

Returns
Dict with added success.

Return type
`dict`

Example:

```
{
    "success": True
}
```

meapi.api.raw.social.**approve_comment_raw**(*client:* Me, *comment_id: int*) → dict

Approve comment.

Parameters

- **client** (*Me*) – *Me* client object.
- **comment_id** (int) – Comment id.

Returns
Dict with comment details.

Return type
dict

Example:

```
{
    'like_count': 0,
    'status': 'approved',
    'message': 'your comment',
    'author': {
        'email': 'exmaple@gmail.com',
        'profile_picture': 'https://d18zp1s.cloudfront.net/b.jpg',
        'first_name': 'Ross',
        'last_name': '',
        'gender': None,
        'uuid': 'ds-dfdf-dcd-af6a-sdffdfdf',
        'is_verified': True,
        'phone_number': 9125342435483,
        'slogan': 'bla bla',
        'is_premium': False,
        'verify_subscription': True
    },
    'is_liked': False,
    'id': 123565,
    'comments_blocked': False
}
```

meapi.api.raw.social.**ask_group_rename_raw**(*client:* Me, *contact_ids: List[int]*, *new_name: str*) → dict

Ask contacts in a group to rename you in their contact book.

Parameters

- **client** (*Me*) – *Me* client object.
- **contact_ids** (List[int]) – list with contacts ids in the same group.
- **new_name** (str) – New name.

Returns
dict with rename success.

Return type
dict

Example:

```
{
    'success': True
}
```

meapi.api.raw.social.**block_comments_raw**(*client:* Me, *uuid: str*) → dict

Block comments from user.

Parameters

- **client** (*Me*) – *Me* client object.
- **uuid** (str) – User uuid.

Returns
Is comments blocked.

Return type
dict

Example:

```
{
    "blocked": true
}
```

meapi.api.raw.social.**delete_comment_raw**(*client:* Me, *comment_id: int*) → dict

Delete comment.

Parameters

- **client** (*Me*) – *Me* client object.
- **comment_id** (int) – Comment id.

Returns
Is comment deleted.

Return type
dict

Example:

```
{
    "success": true
}
```

meapi.api.raw.social.**delete_group_raw**(*client:* Me, *contact_ids: List[int]*) → dict

Delete group.

Parameters

- **client** (*Me*) – *Me* client object.
- **contact_ids** (List[int]) – list with contacts ids in the same group.

Returns
dict with delete success.

Return type
dict

Example:

```
{
    'success': True
}
```

meapi.api.raw.social.**friendship_raw**(*client:* Me, *phone_number:* *int*) → Dict[str, Any]

Get friendship information between you and another number. like count mutual friends, total calls duration, how do you name each other, calls count, your watches, comments, and more.

Parameters

- **client** (*Me*) – *Me* client object.
- **phone_number** (int) – International phone number format.

Returns
Dict with friendship data.

Return type
dict

Example of friendship:

```
{
    "calls_duration": None,
    "he_called": 0,
    "he_named": "He named",
    "he_watched": 3,
    "his_comment": None,
    "i_called": 0,
    "i_named": "You named",
    "i_watched": 2,
    "is_premium": False,
    "mutual_friends_count": 6,
    "my_comment": None,
}
```

meapi.api.raw.social.**get_comment_raw**(*client:* Me, *comment_id:* *int*) → dict

Get comment details, comment text, who and how many liked, create time and more.

Parameters

- **client** (*Me*) – *Me* client object.
- **comment_id** (int) – Comment id.

Returns
Dict with comment details.

Return type
dict

Example:

```
{
    "comment_likes": [
        {
            "author": {
                "email": "yonXXXXXX@gmail.com",
```

(continues on next page)

(continued from previous page)


```
                "first_name": "Jonatan",
                "gender": "M",
                "is_premium": False,
                "is_verified": True,
                "last_name": "Fa",
                "phone_number": 97655764547,
                "profile_picture": "https://d18zaexXXXp1s.cloudfront.net/
↪2eXXefea6dXXXXXXe3.jpg",
                "slogan": None,
                "uuid": "807XXXXX2-414a-b7XXXXX92cd679",
                "verify_subscription": True,
            },
            "created_at": "2022-04-17T16:53:49Z",
            "id": 194404,
        }
    ],
    "like_count": 1,
    "message": "Test comment",
}
```


meapi.api.raw.social.**get_comments_raw**(*client:* Me, *uuid: str*) → dict

Get user comments.

Parameters

- **client** (*Me*) – *Me* client object.
- **uuid** (str) – User uuid.

Returns

Dict with list of comments.

Return type

dict

Example:


```
{
    "comments": [
        {
            "like_count": 2,
            "status": "approved",
            "message": "Test comment",
            "author": {
                "email": "user@domain.com",
                "profile_picture": "https://d18zaexen4dp1s.cloudfront.net/
↪593a9XXXXXXd7437XXXX7.jpg",
                "first_name": "Name test",
                "last_name": "",
                "gender": None,
                "uuid": "8a0XXXXXXXXXXX0a-83XXXXXXb597",
                "is_verified": True,
                "phone_number": 123456789098,
                "slogan": "https://example.com",
                "is_premium": False,
```


(continues on next page)

(continued from previous page)


```
                "verify_subscription": True,
            },
            "is_liked": False,
            "id": 662,
            "comments_blocked": False,
        },
        {
            "like_count": 2,
            "status": "approved",
            "message": "hhaha",
            "author": {
                "email": "haXXXXiel@gmail.com",
                "profile_picture": None,
                "first_name": "Test",
                "last_name": "Test",
                "gender": None,
                "uuid": "59XXXXXXXXXXX-b6c7-f2XXXXXXXXX26d267",
                "is_verified": False,
                "phone_number": 914354653176,
                "slogan": None,
                "is_premium": False,
                "verify_subscription": True,
            },
            "is_liked": True,
            "id": 661,
            "comments_blocked": False,
        },
    ],
    "count": 2,
    "user_comment": None,
}
```


meapi.api.raw.social.**get_deleted_groups_raw**(*client:* Me) → dict

Get group names that you deleted.

Parameters
client (*Me*) – *Me* client object.

Returns
dict with names and contact ids.

Return type
dict

Example:


```
{
    "names": [
        {
            "contact_id": 40108734246,
            "created_at": "2022-04-18T06:08:33Z",
            "hidden_at": "2022-04-23T20:45:19Z",
            "name": "My delivery guy",
            "user": {
```


(continues on next page)

(continued from previous page)


```
                "email": "pnhfdishfois@gmail.com",
                "profile_picture": None,
                "first_name": "Joe",
                "last_name": "",
                "gender": None,
                "uuid": "52XXXXX-b952-XXXX-853e-XXXXXX",
                "is_verified": False,
                "phone_number": 9890987986,
                "slogan": None,
                "is_premium": False,
                "verify_subscription": True,
            },
            "in_contact_list": True,
        }
    ],
    "count": 1,
    "contact_ids": [409879786],
}
```


meapi.api.raw.social.**get_groups_raw**(*client:* Me) → dict

Get groups of names and see how people named you.

Parameters
client (*Me*) – *Me* client object.

Returns
Dict with groups.

Return type
dict

Example:


```
{
    "cached": False,
    "groups": [
        {
            "name": "This is how they name you",
            "count": 1,
            "last_contact_at": "2020-06-09T12:24:51Z",
            "contacts": [
                {
                    "id": 2218840161,
                    "created_at": "2020-06-09T12:24:51Z",
                    "modified_at": "2020-06-09T12:24:51Z",
                    "user": {
                        "profile_picture": "https://XXXXp1s.cloudfront.net/
↪28d5XXX96953feX6.jpg",
                        "first_name": "joz",
                        "last_name": "me",
                        "uuid": "0577XXX-1XXXe-d338XXX74483",
                        "is_verified": False,
                        "phone_number": 954353655531,
                    },
```


(continues on next page)

(continued from previous page)

```
                    "in_contact_list": True,
                }
            ],
            "contact_ids": [2213546561],
        }
    ],
}
```

meapi.api.raw.social.**get_my_social_raw**(*client:* Me) → dict

Get connected social networks to your Me account.

Parameters

- **client** (*Me*) – *Me* client object.
- **uuid** (str) – UUID of the user.

Returns

Dict with social networks and posts.

Return type

dict

Example:

```
{
    "social": {
        "facebook": {
            "posts": [],
            "profile_id": "https://www.facebook.com/app_scoped_user_id/XXXXXXXXXX/
↪",
            "is_active": True,
            "is_hidden": True,
        },
        "fakebook": {
            "is_active": False,
            "is_hidden": True,
            "posts": [],
            "profile_id": None,
        },
        "instagram": {
            "posts": [
                {
                    "posted_at": "2021-12-23T22:21:06Z",
                    "photo": "https://d18zaexen4dp1s.cloudfront.net/XXXXXXXXXXXXX.
↪jpg",
                    "text_first": None,
                    "text_second": "IMAGE",
                    "author": "username",
                    "redirect_id": "CXXXXIz-0",
                    "owner": "username",
                }
            ],
            "profile_id": "username",
            "is_active": True,
```

(continues on next page)

(continued from previous page)


```
            "is_hidden": False,
        },
        "linkedin": {
            "is_active": True,
            "is_hidden": False,
            "posts": [],
            "profile_id": "https://www.linkedin.com/in/username",
        },
        "pinterest": {
            "posts": [],
            "profile_id": "https://pin.it/XXXXXXXX",
            "is_active": True,
            "is_hidden": False,
        },
        "spotify": {
            "is_active": True,
            "is_hidden": False,
            "posts": [
                {
                    "author": "Chandler bing",
                    "owner": "4xgXXXXXXt0pv",
                    "photo": "https://d18zaexen4dp1s.cloudfront.net/
↪9bcXXXfa7dXXXXXXXac.jpg",
                    "posted_at": None,
                    "redirect_id": "4KgES5cs3SnMhuAXuBREW2",
                    "text_first": "My friends playlist songs",
                    "text_second": "157",
                },
                {
                    "author": "Chandler Bing",
                    "owner": "4xgoXcoriuXXXXpt0pv",
                    "photo": "https://d18zaexen4dp1s.cloudfront.net/
↪55d3XXXXXXXXXXXXXXXXXX4.jpg",
                    "posted_at": None,
                    "redirect_id": "3FjSXXXCQPB14Xt",
                    "text_first": "My favorite songs!",
                    "text_second": "272",
                },
            ],
            "profile_id": "4xgot8coriuXXXXXpt0pv",
        },
        "tiktok": {
            "is_active": False,
            "is_hidden": True,
            "posts": [],
            "profile_id": None,
        },
        "twitter": {
            "is_active": True,
            "is_hidden": False,
            "posts": [
                {
```


(continues on next page)

(continued from previous page)

```
                    "author": "username",
                    "owner": "username",
                    "photo": "https://pbs.twimg.com/profile_images/13XXXXX76/
↪AvBXXXX_normal.jpg",
                    "posted_at": "2021-08-24T10:02:45Z",
                    "redirect_id": "https://twitter.com/username/status/1XXXXXX423",
                    "text_first": "My tweet #1 https://t.co/PLXXXX2Tw https://t.co/
↪zXXXXkk",
                    "text_second": None,
                },
                {
                    "author": "username",
                    "owner": "username",
                    "photo": "https://pbs.twimg.com/profile_images/1318XXXX0976/
↪AvBXXXUk_normal.jpg",
                    "posted_at": "2021-08-12T10:09:23Z",
                    "redirect_id": "https://twitter.com/username/status/
↪142XXXXX86624",
                    "text_first": "My second tweet https://t.co/xtqXXXtAC",
                    "text_second": None,
                },
            ],
            "profile_id": "username",
        },
    },
}
```

meapi.api.raw.social.**get_news_raw**(*client:* Me, *os_type: str*) → dict

Get news.

:param client *Me* client object. :type client: *Me* :param os_type: OS type. Can be 'android' or 'ios'. :type os_type: `str` :return: dict with news. :rtype: `dict`

meapi.api.raw.social.**ignore_comment_raw**(*client:* Me, *comment_id: int*) → dict

Ignore comment.

Parameters

- **client** (*Me*) – *Me* client object.
- **comment_id** (`int`) – Comment id.

Returns

Dict with comment details.

Return type

`dict`

Example:

```
{
    'like_count': 0,
    'status': 'ignored',
    'message': 'your comment',
    'author': {
        'email': 'exmaple@gmail.com',
```

(continues on next page)

(continued from previous page)

```
        'profile_picture': 'https://d18zp1s.cloudfront.net/b.jpg',
        'first_name': 'Ross',
        'last_name': '',
        'gender': None,
        'uuid': 'ds-dfdf-dcd-af6a-sdffdfdf',
        'is_verified': True,
        'phone_number': 9125342435483,
        'slogan': 'bla bla',
        'is_premium': False,
        'verify_subscription': True
    },
    'is_liked': False,
    'id': 123565,
    'comments_blocked': False
}
```

meapi.api.raw.social.**like_comment_raw**(*client:* Me, *comment_id:* *int*) → dict

Like comment.

Parameters

- **client** (*Me*) – *Me* client object.
- **comment_id** (int) – Comment id.

Returns
Dict with comment details.

Return type
dict

Example:

```
{
    'id': 12345,
    'created_at': '2022-07-04T21:58:23Z',
    'author': {
        'email': 'user@domain.com',
        'profile_picture': 'https://ehdiued.cloudfront.net/hfidsfds.jpg',
        'first_name': 'Ross',
        'last_name': 'Geller',
        'gender': None,
        'uuid': 'dhius-fdsfs3f7e-fefs-dihoids-odhfods',
        'is_verified': True,
        'phone_number': 9723743824244,
        'slogan': 'Pivot.',
        'is_premium': False,
        'verify_subscription': True
    }
}
```

meapi.api.raw.social.**locations_shared_by_me_raw**(*client:* Me) → List[dict]

Get list of users that you shared your location with them.

Parameters
client (*Me*) – *Me* client object.

Returns
list of dicts with contacts details.

Return type
List[dict]

Example:

```
[
    {
        "first_name": "Rachel Green",
        "last_name": "",
        "phone_number": 1234567890,
        "profile_picture": "https://d18zaexen4dp1s.cloudfront.net/59XXXXXXXXXfa67.
↪jpg",
        "uuid": "XXXXX-XXXXX-XXXX-XXXX-XXXXXX"
    }
]
```

meapi.api.raw.social.**locations_shared_with_me_raw**(*client:* Me) → dict

Get users who have shared a location with you. See also `locations_shared_by_me()`.

Parameters
client (*Me*) – *Me* client object.

Returns
dict with list of uuids and list with users.

Return type
dict

Example:

```
{
    "shared_location_user_uuids": [
        "3850XXX-XXX-XXX-XXX-XXXXX"
    ],
    "shared_location_users": [
        {
            "author": {
                "first_name": "Gunther",
                "last_name": "",
                "phone_number": 3647632874324,
                "profile_picture": "https://d18zaexen4dp1s.cloudfront.net/
↪dXXXXXXXXXXXXXXXXXXb.jpg",
                "uuid": "3850XXX-XXX-XXX-XXX-XXXXX"
            },
            "distance": 1.4099551982832228,
            "i_shared": False
        }
    ]
}
```

meapi.api.raw.social.**numbers_count_raw**(*client:* Me) → dict

Get total count of numbers on Me.

Parameters
client (*Me*) – *Me* client object.

Returns
Dict with numbers count.

Return type
`dict`

Example:

```
{
    'count': 5783726484
}
```

meapi.api.raw.social.**publish_comment_raw**(*client:* Me, *uuid: str*, *your_comment: str*) → dict

Publish comment.

Parameters

- **client** (*Me*) – *Me* client object.
- **uuid** (`str`) – User uuid.
- **your_comment** (`str`) – Comment text.

Returns
Dict with comment details.

Return type
`dict`

Example:

```
{
    'like_count': 0,
    'status': 'waiting',
    'message': 'your comment',
    'author': {
        'email': 'exmaple@gmail.com',
        'profile_picture': 'https://d18zp1s.cloudfront.net/b.jpg',
        'first_name': 'Ross',
        'last_name': '',
        'gender': None,
        'uuid': 'ds-dfdf-dcd-af6a-sdffdfdf',
        'is_verified': True,
        'phone_number': 9125342435483,
        'slogan': 'bla bla',
        'is_premium': False,
        'verify_subscription': True
    },
    'is_liked': False,
    'id': 123565,
    'comments_blocked': False
}
```

meapi.api.raw.social.**remove_social_raw**(*client:* Me, *social_name: str*) → dict

Remove social network from your Me account.

- Available social networks: `facebook`, `instagram`, `spotify`, `twitter`, `tiktok`, `linkedin`, `pintrest`.

Parameters

- **client** (*Me*) – *Me* client object.
- **social_name** (str) – Social network name.

Returns
Dict with removed success.

Return type
dict

Example:

```
{
    "success": True
}
```

meapi.api.raw.social.**report_spam_raw**(*client:* Me, *country_code: str*, *phone_number: str*, *spam_name: str*) → Dict[str, bool]

Report a number as spam.

Parameters

- **client** (*Me*) – *Me* client object.
- **country_code** (str) – Country code.
- **phone_number** (str) – International phone number format.
- **spam_name** (str) – Name of the spammer.

Returns
Dict with spam report success.

Return type
dict

Example of results::
{'success': True}

meapi.api.raw.social.**restore_group_raw**(*client:* Me, *contact_ids: List[int]*) → dict

Restore group.

Parameters

- **client** (*Me*) – *Me* client object.
- **contact_ids** (List[int]) – list with contacts ids in the same group.

Returns
dict with restore success.

Return type
dict

Example:

```
{
    'success': True
}
```

meapi.api.raw.social.**share_location_raw**(*client:* Me, *uuid: str*)

Share your location with user.

Parameters

- **client** (*Me*) – *Me* client object.
- **uuid** (str) – UUID of the user that you want to share the location with him.

Returns
Dict with shared location success.

Return type
dict

Example:

```
{
    'success': True
}
```

meapi.api.raw.social.**stop_shared_locations_raw**(*client:* Me, *uuids: List[str]*) → dict

Stop location that shared with you.

Parameters

- **client** (*Me*) – *Me* client object.
- **uuids** (List[str]) – List of users UUID's that you want get their location.

Returns
Dict with stopped shared location success.

Return type
dict

Example:

```
{
    'success': True
}
```

meapi.api.raw.social.**stop_sharing_location_raw**(*client:* Me, *uuids: List[str]*) → dict

Stop sharing your location with user.

Parameters

- **client** (*Me*) – *Me* client object.
- **uuids** (List[str]) – List of users UUID's that you want to stop sharing the location with them.

Returns
Dict with stopped sharing location success.

Return type
dict

Example:

```
{
    'success': True
}
```

meapi.api.raw.social.**suggest_turn_on_comments_raw**(*client:* Me, *uuid:* *str*)

Ask from user to turn on comments in his profile.

Parameters

- **client** (*Me*) – *Me* client object.
- **uuid** (str) – UUID of the user.

Returns
Dict with turned on comments success.

Return type
dict

Example:

```
{
    'requested': True
}
```

meapi.api.raw.social.**suggest_turn_on_location_raw**(*client:* Me, *uuid:* *str*)

Ask from user to share his location with you.

Parameters

- **client** (*Me*) – *Me* client object.
- **uuid** (str) – UUID of the user.

Returns
Dict with requested location success.

Return type
dict

Example:

```
{
    'requested': True
}
```

meapi.api.raw.social.**suggest_turn_on_mutual_raw**(*client:* Me, *uuid:* *str*)

Ask from user to turn on mutual contacts in his profile.

Parameters

- **client** (*Me*) – *Me* client object.
- **uuid** (str) – UUID of the user.

Returns
Dict with turned on mutual success.

Return type
dict

Example:

```
{
    'requested': True
}
```

meapi.api.raw.social.**switch_social_status_raw**(*client:* Me, *social_name: str*) → dict

Switch social network status (hidden or shown) from your Me account.

- Available social networks: `facebook`, `instagram`, `spotify`, `twitter`, `tiktok`, `linkedin`, `pintrest`.

Parameters

- **client** (*Me*) – *Me* client object.
- **social_name** (`str`) – Social network name.

Returns
Dict with switched success.

Return type
`dict`

Example:

```
{
    'is_hidden': False
}
```

meapi.api.raw.social.**unlike_comment_raw**(*client:* Me, *comment_id: int*) → dict

Unlike comment.

Parameters

- **client** (*Me*) – *Me* client object.
- **comment_id** (`int`) – Comment id.

Returns
Dict with comment details.

Return type
`dict`

Example:

```
{
    'like_count': 0,
    'status': 'approved',
    'message': 'Test',
    'author': {
        'email': 'user@domain.com',
        'profile_picture': 'https://hiiu.cloudfront.net/hdiufds.jpg',
        'first_name': 'Monica',
        'last_name': '',
        'gender': None,
        'uuid': 'djhids-oiehda-huds-dhcuds-dhfidsdsf',
        'is_verified': True,
        'phone_number': 973437824255,
        'slogan': 'I know!',
        'is_premium': False,
        'verify_subscription': True
    },
```

(continues on next page)

(continued from previous page)

```
    'is_liked': False,
    'id': 123456,
    'comments_blocked': False
}
```

meapi.api.raw.social.**update_location_raw**(*client:* Me, *latitude:* *float*, *longitude:* *float*)

Update your location.

Parameters

- **client** (*Me*) – *Me* client object.
- **latitude** (float) – Latitude.
- **longitude** (float) – Longitude.

Returns
Dict with updated location success.

Return type
dict

Example:

```
{
    'success': True
}
```

meapi.api.raw.social.**who_deleted_raw**(*client:* Me) → List[dict]

Get a list of users that deleted you from their contacts.

Parameters
client (*Me*) – *Me* client object.

Returns
List of dicts with users.

Return type
List[dict]

Example of results:

```
[
    {
        "created_at": "2021-09-12T15:42:57Z",
        "user": {
            "email": "",
            "profile_picture": None,
            "first_name": "Test",
            "last_name": "Test",
            "gender": None,
            "uuid": "aa221ae8-XXX-4679-XXX-91307XXX5a9a2",
            "is_verified": False,
            "phone_number": 123456789012,
            "slogan": None,
            "is_premium": False,
            "verify_subscription": True,
```

(continues on next page)

(continued from previous page)

```
        },
    }
]
```

meapi.api.raw.social.**who_watched_raw**(*client:* Me) → List[dict]

Get a list of users that watched you.

Parameters
client (*Me*) – *Me* client object.

Returns
List of dicts with users.

Return type
List[dict]

Example of results:

```
[
    {
        "last_view": "2022-04-16T17:13:24Z",
        "user": {
            "email": "eliezXXXXXXXXX94@gmail.com",
            "profile_picture": "https://d18zXXXXXXXXXXXXXcb14529ccc7db.jpg",
            "first_name": "Test",
            "last_name": None,
            "gender": None,
            "uuid": "f8d03XXX97b-ae86-35XXXX9c6e5",
            "is_verified": False,
            "phone_number": 97876453245,
            "slogan": None,
            "is_premium": True,
            "verify_subscription": True,
        },
        "count": 14,
        "is_search": None,
    }
]
```

Settings

meapi.api.raw.settings.**change_settings_raw**(*client:* Me, ***kwargs*) → dict

Change current settings.

Parameters
kwargs – Dict with new settings.

Returns
Dict with settings.

Return type
dict

Example:

```
{
    "birthday_notification_enabled": True,
    "comments_enabled": True,
    "comments_notification_enabled": True,
    "contact_suspended": False,
    "distance_notification_enabled": True,
    "language": "iw",
    "last_backup_at": None,
    "last_restore_at": None,
    "location_enabled": True,
    "mutual_contacts_available": True,
    "names_notification_enabled": True,
    "notifications_enabled": True,
    "system_notification_enabled": True,
    "who_deleted_enabled": True,
    "who_deleted_notification_enabled": True,
    "who_watched_enabled": True,
    "who_watched_notification_enabled": True,
}
```


meapi.api.raw.settings.**get_settings_raw**(*client:* Me) → dict

Get current settings.

Returns
Dict with settings.

Return type
dict

Example:


```
{
    "birthday_notification_enabled": True,
    "comments_enabled": True,
    "comments_notification_enabled": True,
    "contact_suspended": False,
    "distance_notification_enabled": True,
    "language": "iw",
    "last_backup_at": None,
    "last_restore_at": None,
    "location_enabled": True,
    "mutual_contacts_available": True,
    "names_notification_enabled": True,
    "notifications_enabled": True,
    "spammers_count": 24615,
    "system_notification_enabled": True,
    "who_deleted_enabled": True,
    "who_deleted_notification_enabled": True,
    "who_watched_enabled": True,
    "who_watched_notification_enabled": True,
}
```

Notifications

meapi.api.raw.notifications.**get_notifications_raw**(*client:* Me, *page_number:* *int*, *results_limit:* *int*, *categories:* *Optional[List[str]]* = *None*) → dict

Get notifications.

Parameters

- **page_number** (int) – Page number.
- **results_limit** (int) – Number of results per page.
- **categories** (List[str]) – List of categories to filter.

Returns
Dict with notifications.

Return type
dict

Example:


```
{
    "count": 94,
    "next": "https://app.mobile.me.app/notification/notification/items/?page=2&page_
↪size=30&status=distributed",
    "previous": None,
    "results": [
        {
            "id": 103466332,
            "created_at": "2022-03-18T11:17:09Z",
            "modified_at": "2022-03-18T11:17:09Z",
            "is_read": False,
            "sender": "2e7XXX-84XXXX-4ec7-b6cb-d4XXXXXX",
            "status": "distributed",
            "delivery_method": "push",
            "distribution_date": "2022-03-18T11:17:09Z",
            "message_subject": None,
            "message_category": "BIRTHDAY",
            "message_body": None,
            "message_lang": "iw",
            "context": {
                "name": "Ross geller",
                "uuid": "2e7XXXX-XXXX-XXXX-b6cXb-d46XXXXX1",
                "category": "BIRTHDAY",
                "phone_number": 97849743536,
                "notification_id": None,
                "profile_picture": None,
            },
        },
        {
            "id": 18987495325,
            "created_at": "2022-04-06T11:18:03Z",
            "modified_at": "2022-04-06T11:18:03Z",
            "is_read": False,
            "sender": "5XXXXX0e-XXXX-XXXX-XXXX-XXXXXXX",
```


(continues on next page)

(continued from previous page)


```
            "status": "distributed",
            "delivery_method": "push",
            "distribution_date": "2022-04-06T11:18:03Z",
            "message_subject": None,
            "message_category": "UPDATED_CONTACT",
            "message_body": None,
            "message_lang": "iw",
            "context": {
                "name": "Chandler",
                "uuid": "XXXXXX-XXXX-XXXXX-XXX-XXXXX",
                "category": "UPDATED_CONTACT",
                "new_name": "Your new name",
                "phone_number": 8479843759435,
                "notification_id": None,
                "profile_picture": None,
            },
        },
        {
            "id": 17983743351,
            "created_at": "2022-04-11T06:45:27Z",
            "modified_at": "2022-04-11T06:45:27Z",
            "is_read": False,
            "sender": "XXXXXX-XXXX-XXXXX-XXX-XXXXX",
            "status": "distributed",
            "delivery_method": "push",
            "distribution_date": "2022-04-11T06:45:27Z",
            "message_subject": None,
            "message_category": "CONTACT_ADD",
            "message_body": None,
            "message_lang": "iw",
            "context": {
                "name": "Monica",
                "uuid": "XXXXXX-XXXX-XXXXX-XXX-XXXXX",
                "category": "CONTACT_ADD",
                "new_name": "Ross",
                "phone_number": 878634535436,
                "notification_id": None,
                "profile_picture": None,
            }
        }
    ]
}
```


meapi.api.raw.notifications.**read_notification_raw**(*client:* Me, *notification_id:* *int*) → dict

Mark notification as read.

Parameters
notification_id (int) – Notification ID.

Returns
Dict with notification.

Return type
dict

Example:

```
{
    'id': 3487438454,
    'created_at': '2022-06-28T22:12:36Z',
    'modified_at': '2022-06-30T23:56:59Z',
    'is_read': True,
    'sender': 'fejdsfns-hdiu-ddv-83c1-hdisds',
    'status': 'distributed',
    'delivery_method': 'push',
    'distribution_date': '2022-06-28T22:12:35Z',
    'message_subject': None,
    'message_category': 'NEW_COMMENT',
    'message_body': None,
    'message_lang': 'en',
    'context': {
        'name': 'Monica',
        'uuid': 'dsfsds-dsfd-490a-dfdfg-dfoeiffs',
        'category': 'NEW_COMMENT',
        'phone_number': 972538607327,
        'notification_id': None,
        'profile_picture': 'https://dfsfs.cloudfront.net/dhiucds.jpg'
    }
}
```

meapi.api.raw.notifications.**unread_notifications_count_raw**(*client:* Me) → dict

Get number of unread notifications.

Returns
Dict with number of unread notifications.

Return type
dict

7.2.3 Models

Base model

class meapi.models.me_model.**MeModel**

Base class for all models.

- Allow instances to be comparable, subscript, hashable, immutable (In some cases), and json serializable.

Examples

```
>>> my_profile = me.get_my_profile() # Get your profile.
>>> my_profile.name # regular access
>>> my_profile['name'] # subscript access
>>> my_profile.get('name', default='') # safe access
>>> my_profile.as_dict() # get all data as dict
>>> my_profile.as_json() # get all data as json string
>>> my_profile == other_profile # compare two objects
```

Methods:

as_dict(*only: Optional[Union[str, List[str]]] = None*, *exclude: Optional[Union[str, List[str]]] = None*, *recursive: bool = True*) → Dict[str, Any]

Return class data as `dict`.

Examples

```
>>> my_profile = me.get_my_profile() # Get your profile.
>>> my_profile.as_dict()
{'phone_number': 972123456789, 'first_name': 'David', 'last_name': 'Lev',
↪'socials': {'lin ...}
>>> my_profile.as_dict(only=('phone_number', 'first_name'), recursive=False)
{'phone_number': 972123456789, 'first_name': 'David'}
```

Parameters

- **only** (`str | list[str]`) – Return only the given keys.
- **exclude** (`str | list[str]`) – Exclude the given keys.
- **recursive** (`bool`) – If `True` (default), return all nested objects as dict.

as_json(*only: Optional[Union[str, List[str]]] = None*, *exclude: Optional[Union[str, List[str]]] = None*, *recursive: bool = True*, *ensure_ascii: bool = False*) → str

Return class data in `json` format.

Parameters

- **only** (`str | list[str]`) – Return only the given keys.
- **exclude** (`str | list[str]`) – Exclude the given keys.
- **recursive** (`bool`) – If `True` (default), return all nested objects as dict.
- **ensure_ascii** (`bool`) – If `True`, all non-ASCII characters in the output are escaped with `\uXXXX` sequences.

get(*item: str*, *default: Optional[Any] = None*)

Return the value of the attribute with the given name.

- **Example:**

```
>>> my_profile = me.get_my_profile() # Get your profile.
>>> my_profile.get('phone_number')
```

Parameters

- **item** (str) – The name of the attribute.
- **default** (any) –

  The default value to return if the attribute does not exist.

  - Default: None

Returns

The value of the attribute, or default if the attribute does not exist.

Profile model

class meapi.models.profile.**Profile**

Represents the user's profile. can also be used to update you profile details.

- Modifiable attributes are marked with modifiable.
- For more information about the modifiable attributes, see *update_profile_details()*.

Example

```
>>> # Update your profile details.
>>> my_profile = me.get_my_profile()
>>> my_profile.name = "Chandler Bing"
>>> my_profile.date_of_birth = "1968-04-08"
>>> my_profile.slogan = "Hi, I'm Chandler. I make jokes when I'm uncomfortable."
>>> my_profile.profile_picture = "/home/chandler/Downloads/my_profile.jpg"
```

Parameters

- **name** (str *optional*, *modifiable*) – The user's full name (first_name + last_name).
- **first_name** (str *optional*, *modifiable*) – The user's first name.
- **last_name** (str *optional*, *modifiable*) – The user's last name.
- **profile_picture** (str *optional*, *modifiable*) – The user's profile picture url.
- **slogan** (str *optional*, *modifiable*) – The user's bio.
- **email** (str *optional*, *modifiable*) – The user's email.
- **gender** (str *optional*, *modifiable*) – The user's gender: M for male and F for female.
- **date_of_birth** (date *optional*, *modifiable*) – The user's date of birth.
- **age** (int) – The user's age. calculated from date_of_birth if exists, else 0.
- **social** (*Social optional*) – The user's social media networks.
- **phone_number** (int) – The user's phone number.
- **uuid** (str) – The user's unique ID.
- **phone_prefix** (str) – The user's phone prefix.
- **device_type** (str *optional*, *modifiable*) – The user's device type: android or ios.
- **login_type** (str *optional*, *modifiable*) – The user's login type: email or apple.

- **who_deleted_enabled** (bool) – Whether the user can see who deleted him (Only if is_premium, Or if he uses meapi ;).
- **who_deleted** (List[*Deleter*] *optional*) – The users who deleted him.
- **who_watched_enabled** (bool) – Whether the user can see who watched his profile (Only if is_premium, Or if he uses meapi ;).
- **who_watched** (List[*Watcher*] *optional*) – The users who watched him.
- **friends_distance** (List[*User*] *optional*) – The users who shared their location with you.
- **carrier** (str *optional*, *modifiable*) – The user's cell phone carrier.
- **comments_enabled** (bool) – Whether the user is allowing comments. You can ask the user to turn on comments with *suggest_turn_on_comments()*.
- **comments_blocked** (bool) – Whether the user blocked you from commenting on his profile.
- **country_code** (str) – The user's two-letter country code.
- **location_enabled** (bool *optional*) – Whether the user is allowing location.
- **is_shared_location** (bool *optional*) – Whether the user is sharing their location with you. You can ask the user to share his location with *suggest_turn_on_location()*.
- **share_location** (bool *optional*) – Whether the user is sharing their location with you.
- **distance** (float *optional*) – The user's distance from you.
- **location_longitude** (float *optional*) – The user's location longitude coordinate.
- **location_latitude** (float *optional*) – The user's location latitude coordinate.
- **location_name** (str *optional*, *modifiable*) – The user's location name.
- **is_he_blocked_me** (bool *optional*) –

  Whether the user has blocked you from seeing his profile (me_full_block, See *block_profile()*).

  - If True, this profile will contain only the basic information, like name, uuid and phone_number.
- **is_permanent** (bool *optional*) – Whether the user is permanent.
- **mutual_contacts_available** (bool *optional*) – Whether the user has mutual contacts available. You can ask the user to turn on this feature with *suggest_turn_on_mutual()*.
- **mutual_contacts** (List[*MutualContact*] *optional*) – For more information about mutual contacts.
- **is_premium** (bool) – Whether the user is a premium user.
- **is_verified** (bool) – Whether the user is verified.
- **gdpr_consent** (bool) – Whether the user has given consent to the GDPR.
- **facebook_url** (str *optional*, *modifiable*) – The user's Facebook ID.
- **google_url** (str *optional*, *modifiable*) – The user's Google ID.
- **me_in_contacts** (bool *optional*) – Whether you are in the user's contacts.
- **user_type** (str *optional*) –

  The user's type: the color of the user in the app:

- BLUE: Verified Caller ID from ME users (100% ID).
- GREEN: Identified call with a very reliable result.
- YELLOW: Uncertain Identification (Unverified).
- ORANGE: No identification (can be reported).
- RED: Spam calls.

- **verify_subscription** (bool *optional*) – Whether the user has verified their subscription.

Get friendship

friendship().

Get comments

get_comments().

Get the profile as Vcard

as_vcard().

Block the profile

block().

Unblock this profile

unblock().

Report this profile as spam

report_spam().

User model

class meapi.models.user.**User**

Represents a user.

- A user is a person who log in to the app.
- If you search a phone number with *phone_search()*, you will get a contact, but if this contact registered on the app, you get a user attribute.

Parameters

- **name** (str) – The fullname of the user. combined with first_name and last_name.
- **first_name** (str) – The first name of the user.
- **last_name** (str) – The last name of the user.
- **uuid** (str) – The unique identifier of the user. can be used to perform actions on the user.
- **phone_number** (int) – The phone number of the user.
- **gender** (str *optional*) – The gender of the user. M for male, F for female, and None if the user didn't specify.
- **email** (str *optional*) – The email of the user.
- **profile_picture** (str *optional*) – Url to profile picture of the user.
- **slogan** (str *optional*) – The bio of the user.

- **is_verified** (bool *optional*) – Whether the user is verified (Has at least two social connected accounts).
- **is_premium** – Whether the user is paying for premium features (Like the ability to use who watch his profile, who deleted him from his contacts, no ads, and more).

Get friendship

friendship().

Get comments

get_comments().

Get the user as Vcard

as_vcard().

Block the user

block().

Unblock this user

unblock().

Report this user as spam

report_spam().

Contact model

class meapi.models.contact.**Contact**

Represents a contact.

Parameters

- **name** (str) – The name of the contact.
- **phone_number** (int) – The phone number of the contact.
- **id** (int) – The id of the contact.
- **picture** (str *optional*) – The url picture of the contact.
- **user** (*User* *optional*) – The user of the contact. if the user register on the app.
- **suggested_as_spam** (int *optional*) – The number of times the contact has been suggested as spam.
- **user_type** (str *optional*) –

  The user's type: the color of the user in the app:

  - BLUE: Verified Caller ID from ME users (100% ID).
  - GREEN: Identified call with a very reliable result.
  - YELLOW: Uncertain Identification (Unverified).
  - ORANGE: No identification (can be reported).
  - RED: Spam calls.
- **is_permanent** (bool *optional*) – Whether the contact is permanent.
- **is_pending_name_change** (bool *optional*) – Whether the contact is pending name change.
- **cached** (bool *optional*) – Whether the results from the api is cached.

- **is_shared_location** (bool *optional*) – Whether the contact is shared location.
- **created_at** (datetime *optional*) – The date of the contact creation.
- **modified_at** (datetime *optional*) – The date of the contact modification.
- **in_contact_list** (bool *optional*) – Whether the contact is in the contact list.
- **is_my_contact** (bool *optional*) – Whether the contact is my contact.

Get friendship

friendship().

Get comments

get_comments().

Get the contact as Vcard

as_vcard().

Block this contact

block().

Unblock this contact

unblock().

Report this contact as spam

report_spam().

Common model

class meapi.models.common.**_CommonMethodsForUserContactProfile**

Common methods for *Profile*, *User* and *Contact*.

as_vcard(*prefix_name: str = ''*, *dl_profile_picture: bool = False*, ***kwargs*) → str

Get contact data in vcard format in order to add it to your contacts book.

Usage examples:

```
# Get your profile as vcard
my_profile = me.get_my_profile()
print(my_profile.as_vcard(twitter='social.twitter.profile_id', gender=
↪'gender')

# Save profiles as vcard file
uuids = ['xx-xx-xx-xx', 'yy-yy-yy-yy', 'zz-zz-zz-zz']
profiles = [me.get_profile(uuid) for uuid in uuids] # can raise rate limit␣
↪exception.
vcards = [profile.as_vcard(prefix_name="Imported", dl_profile_
↪picture=False,
    location='location_name') for profile in profiles]
with open('contacts.vcf', 'w') as contacts:
    contacts.write('\n'.join(vcards))
```

Parameters

- **prefix_name** – (str): If you want to add prefix to the name of the contact, like `Mr.`, `Mrs.`, `Imported` etc. *Default:* empty string `""`.

- **dl_profile_picture** – (bool): If you want to download and add profile picture to the vcard (if available). *Default:* `False`.
- **kwargs** –

  Add any other data to the `notes` field of the vcard. The key must be, of course, exists in the object as attr eith value of `str` or `int`.

  - For example, if you want to add a gender information to the contact, you can pass the parameter `gender='gender'`
  - The key uses as the title in the notes (you name it as you like), and the value is the attribute name of the object.
  - You can go even deeper: if Profile object provided, you may want to do something like `twitter='social.twitter.profile_id'`.
  - No exception will be raised if the key doesn't exist.

Returns
Vcard format as string. See Wikipedia for more information.

Return type
`str`

Results example:

```
BEGIN:VCARD
VERSION:3.0
FN;CHARSET=UTF-8;ENCODING=QUOTED-PRINTABLE:Rachel Green
TEL;CELL:1234567890
PHOTO;ENCODING=BASE64;JPEG:/9j/4AAQSgyIR..........
EMAIL:rachelg@friends.tv
BDAY:1969-05-05
NOTE;CHARSET=UTF-8;ENCODING=QUOTED-PRINTABLE:Twitter: RachelGreeen | Gender: F
END:VCARD
```

block(*block_contact=True*, *me_full_block=True*) → bool

Block a contact.

Parameters

- **block_contact** – (bool): If you want to block the contact from calls. *Default:* `True`.
- **me_full_block** – (bool): If you want to block the contact from Me platform. *Default:* `True`.

Returns
`True` if the contact was blocked successfully, else `False`.

Return type
`bool`

Raises
TypeError – If you try to block yourself.

friendship() → *Friendship*

Returns the friendship status of the contact.

Returns
`True` if the contact is your friend, else `False`.

Return type
bool

`get_comments`() → List[*Comment*]

Returns the comments of the contact.

Returns
The comments of the contact.

Return type
List[*Comment*]

`get_profile`() → Optional[*Profile*]

Returns the profile of the contact.

Returns
The profile of the contact or `None` if the contact has no user.

Return type
Profile | None

`report_spam`(*spam_name: str*, *country_code: str*) → bool

Report this contact as spam.

- The same as *report_spam()*.

Parameters

- **spam_name** – (str): Name of the spammer.
- **country_code** – (str): Country code of the spammer.

Returns
`True` if the contact was reported successfully, else `False`.

Return type
bool

`unblock`(*unblock_contact=True*, *me_full_unblock=True*) → bool

Unblock a contact.

Parameters

- **unblock_contact** – (bool): If you want to unblock the contact from calls. *Default:* `True`.
- **me_full_unblock** – (bool): If you want to unblock the contact from Me platform. *Default:* `True`.

Returns
`True` if the contact was unblocked successfully, else `False`.

Return type
bool

Raises
TypeError – If you try to unblock yourself.

Group model

class meapi.models.group.**Group**

Represents a group of users that save you in their contact list in the same name

- For more information about this feature

Examples

```
>>> my_groups = me.get_groups()
>>> group = my_groups[0]
>>> group.name
'Phoebe Buffay'
>>> group.count
6
>>> group.last_contact_at
datetime.datetime(2004, 5, 6, 21, 0)
>>> group.ask_to_rename(new_name='Regina Phalange')
```

Parameters

- **name** (str) – The name of the group, how your number saved in their contact list.
- **count** (int) – The number of users in the group.
- **last_contact_at** (datetime *optional*) – The last time that you saved in someone's contact list.
- **contacts** (List[*User*]) – The users that are in the group.
- **contact_ids** (List[int]) – The ids of the users that are in the group.
- **is_active** (bool) – Is the group active. - You can use *delete()* to hide the group and *restore()* to restore it.

Methods:

delete() → bool

Deletes the group.

- The same as *delete_group()*.
- You get True even if the group is already hidden.

Returns
True if the group was deleted, False otherwise.

Return type
bool

restore() → bool

Restores the group.

- The same as *restore_group()*.
- You get True even if the group is already active.

Returns
`True` if the group was restored, `False` otherwise.

Return type
`bool`

ask_to_rename(*new_name*) → bool

Asks from the users in the group to rename you in their contact list.

- The same as *ask_group_rename()*.
- You can't adk rename a group if it's hidden (`is_active=False`).

Parameters
new_name (`str`) – The new name that you want them to rename you in their contact list.

Returns
`True` if the suggested send, `False` otherwise.

Return type
`bool`

Social model

class meapi.models.social.**Social**

Represents user's social media accounts.

Examples

```
>>> my_socials = me.get_socials()
>>> my_socials.instagram.add(token_or_url="xxxxxxxxxxxxxx")
True
>>> my_socials.instagram.profile_url
https://instagram.com/courteneycoxofficial
>>> my_socials.instagram.posts[0].text
"Okay, hypothetically, why won't I be married when I'm 40?"
```

Parameters

- **facebook** (SocialMediaAccount) – Facebook account.
- **fakebook** (SocialMediaAccount) – Fakebook account.
- **instagram** (SocialMediaAccount) – Instagram account.
- **linkedin** (SocialMediaAccount) – LinkedIn account.
- **pinterest** (SocialMediaAccount) – Pinterest account.
- **spotify** (SocialMediaAccount) – Spotify account.
- **tiktok** (SocialMediaAccount) – Tiktok account.
- **twitter** (SocialMediaAccount) – Twitter account.

class meapi.models.social.**SocialMediaAccount**

Represents user's social media account.

Examples

```
>>> my_socials = me.get_socials()
>>> my_socials.spotify.add(token_or_url="xxxxxxxxxxxxxx") # connect spotify account
True
>>> my_socials.spotify.hide() # hide account from public
True
>>> my_socials.spotify.unhide() # unhide account from public
True
>>> my_socials.spotify.remove() # remove account
```

Parameters

- **name** (str) – Name of social media account.
- **profile_id** (str *optional*) – Profile ID or username of social media account.
- **profile_url** (str *optional*) – The profile url of social media account.
- **posts** (List[*Post*]) – List of posts from social media account.
- **is_active** (bool) – Is social media account active.
- **is_hidden** (bool) – Is social media account hidden by the user (You can see it because the API sends it anyway ;).

Methods:

add(*token_or_url: str*) → bool

Add social media account to your Me profile.

- If you have at least two social media accounts, you get verification tag on your profile. is_verified = True.

Parameters

token_or_url (str) –

Token or URL of social media account.

- See *add_social()* for more information.

Returns

True if successfully added, False otherwise.

Return type

bool

remove() → bool

Remove social media account from your Me profile.

Returns

True if successfully removed, False otherwise.

- You get True even if the social media account not active.

Return type

bool

hide() → bool

Hide social media account in your Me profile.

- You get `True` even if the social media account not active or already hidden.

Returns
`True` if successfully hidden, `False` otherwise.

Return type
`bool`

unhide() → bool

Unhide social media account in your Me profile.

- You get `True` even if the social media already unhidden.

Returns
`True` if successfully unhidden, `False` otherwise.

Return type
`bool`

class meapi.models.social.**Post**

Represents Social Media post.

- Not every social media account has posts.

Parameters

- **author** (`str` *optional*) – Author of post.
- **owner** (`str` *optional*) – Owner of post.
- **text_first** (`str` *optional*) – First text of post.
- **text_second** (`str` *optional*) – Second text of post.
- **redirect_id** (`str` *optional*) – Redirect ID of post.
- **photo** (`str` *optional*) – Photo of post.

Comment model

class meapi.models.comment.**Comment**

Represents a comment.

Examples

```
>>> my_comments = me.get_comments()
>>> my_comments[0].message
'We were on a break!'
>>> my_comments[0].like_count
7
>>> my_comments[0].author.name
'Ross Geller'
>>> my_comments[0].like()
True
>>> my_comments[0].reply("I got off the plane.")
<Comment id=123 status=waiting msg=I got off the plane. author=Rachel Green>
```

Parameters

- **message** (str) – The message of the comment.
- **id** (int) – The id of the comment.
- **status** (str) – The status of the comment: approved, ignored, waiting, deleted.
- **author** (*User*) – The creator of the comment.
- **like_count** (int) – The number of likes of the comment.
- **comment_likes** (list of *User*) – The list of users who liked the comment.
- **created_at** (datetime | None) – The date of the comment creation.
- **is_liked** (bool) – Whether the creator liked his comment.
- **comments_blocked** (bool) – Whether the user blocked the comments of the comment.

Methods:

approve() → bool

Approve the comment.

- You can only approve comments that posted by others on your own profile.
- The same as *approve_comment()*.

Returns
Is approve success.

Return type
bool

edit(*new_msg: str*, *remove_credit: bool = False*) → bool

Edit the comment.

- You can only edit comments that posted by you.
- The same as *publish_comment()*.

Parameters

- **new_msg** (str) – The new message of the comment.

- **remove_credit** (bool) – Whether to remove the credit to meapi from the comment.

Returns
Is edit success.

Return type
bool

ignore() → bool

Ignore and hide the comment.

- You can only ignore and hide comments that posted by others on your own profile.
- The same as *ignore_comment()*.

Returns
Is ignore success.

Return type
bool

delete() → bool

Delete the comment.

- You can only delete comments that posted on your own profile or by you.
- The same as *delete_comment()*.

Returns
Is delete success.

Return type
bool

like() → bool

Like the comment.

- The same as *like_comment()*.

Returns
Is like success.

Return type
bool

unlike() → bool

Unlike the comment.

- The same as *unlike_comment()*.

Returns
Is unlike success.

Return type
bool

reply(*your_comment: str*) → Optional[*Comment*]

Publish a comment in the profile of the comment author.

- The same as *publish_comment()*.
- if you already replied to the comment, this will edit the comment like *edit()*.

Parameters
your_comment (str) – The message of the comment.

Returns
The new comment.

Return type
Comment

block()

Block the author of the comment from posting comments on your profile.

- The same as *block_comments()*.
- This will not delete the comment. It will just block the author from editing or posting comments on your profile.

Returns
Is block success.

Return type
bool

Watcher model

class meapi.models.watcher.**Watcher**

Represents a Watcher, user who watch your profile.

- For more information about Watcher

Examples

```
>>> my_watchers = me.who_watched()
>>> watcher = my_watchers[0]
>>> watcher.user.name
'Mike Hannigan'
>>> watcher.count
15
>>> me.publish_comment(uuid=watcher.user,your_comment="So, what are your intentions␣
↪with my Phoebe?")
<Comment id=321 status=waiting msg=o, what are your intentions with my Phoebe?␣
↪author=Joey Tribbiani>
```

Parameters

- **last_view** (datetime) – Date of last view.
- **user** (*User*) – The user who watch your profile.

- **count** (int) – The number of views.
- **is_search** (bool) – Whether the user is searching your profile.

Deleter model

class meapi.models.deleter.**Deleter**

Represents a Deleter, user who delete you from his contacts.

- For more information about Deleter

Examples

```
>>> my_deleters = me.who_deleted()
>>> deleter = my_deleters[0]
>>> deleter.user.name
'Janine Lecroix'
>>> me.publish_comment(uuid=deleter.user,your_comment="How You Doin'?")
<Comment id=456 status=waiting msg=How You Doin'? author=Joey Tribbiani>
```

Parameters

- **created_at** (str) – Date of delete.
- **user** (*User*) – User who delete you.

Friendship model

class meapi.models.friendship.**Friendship**

Represents a Friendship.

- Friendship is a relationship between you and another user.
- For more information about Friendship

Examples

```
>>> janice_and_i = me.friendship(phone_number=19690300000000)
>>> janice_and_i.he_named
'Little bingaling'
>>> janice_and_i.i_named
'Oh. My. God.'
>>> janice_and_i.his_comment
'You're my little love muffin'
>>> janice_and_i.my_comment
'I seek you out!
```

Parameters

- **calls_duration** (int) – The duration of your calls in seconds.
- **he_called** (int) – The number of times the other user has called you.

- **i_called** (int) – The number of times you have called the other user.
- **he_named** (str) – How the other user named you in his contacts book.
- **i_named** (str) – How you named the other user in your contacts book.
- **he_watched** (int) – The number of times the other user has watched your profile.
- **his_comment** (str) – The comment the other user has comment on your profile.
- **my_comment** (str *optional*) – The comment you have comment on the other user's profile.
- **i_watched** (int) – The number of times you have watched the other user's profile.
- **mutual_friends_count** (int) – The number of mutual contacts between you and the other user.
- **is_premium** (bool) – Whether the other user is a premium user.

Notification model

class meapi.models.notification.**Notification**

Represents a Notification from the app.

- Notification could be new comment, new profile watch, new deleted, contact birthday, suggestion, etc.
- For more information about Notification

Examples

```
>>> my_notifications = me.get_notifications()
>>> notification = my_notifications[0][0]
>>> notification.category
'UPDATED_CONTACT'
>>> notification.name
'Mike Hannigan'
>>> notification.new_name
'Princess Consuela Banana-Hammock'
>>> me.publish_comment(uuid=notification.uuid,your_comment="Hi *** bag!")
<Comment id=678 status=waiting msg=Hi *** bag! author=Phoebe Buffay>
```

Parameters

- **id** (int) – The id of the notification.
- **created_at** (*datetime* `) – Date of creation.
- **modified_at** (*datetime* `) – Date of last modification.
- **is_read** (bool) – Whether the notification is read.
- **sender** (str) – UUID of the sender of the notification.
- **status** (str) – Status of the notification.
- **delivery_method** (str) – Delivery method of the notification. Most likely push.
- **distribution_date** (datetime) – Date of distribution.

- **category** (str) – Category of the notification.
- **message_lang** (str) – Language of the notification, en, he etc.
- **message_subject** (str *optional*) – Subject of the notification.
- **message_body** (str *optional*) – Body of the notification.
- **context** (dict *optional*) – The context of the notification: name, uuid, new_name, tag, profile_picture and more.

Methods:

read() → bool

Mark the notification as read.

- The same as *read_notification()*.

Returns
Whether the notification was marked as read.

Return type
bool

Settings model

class meapi.models.settings.**Settings**

Manage your social, notification and app settings.

- You can edit your settings by simply assigning a new value to the attribute.
- Modifiable attributes are marked with modifiable.
- For more information about the modifiable attributes, see *change_settings()*.

Example

```
>>> # Take control of your privacy:
>>> my_settings = me.get_settings()
>>> my_settings.who_watched_enabled = False
>>> my_settings.who_deleted_enabled = False
>>> my_settings.mutual_contacts_available = False
>>> my_settings.comments_enabled = False
>>> my_settings.location_enabled = False
```

Parameters

- **who_deleted_enabled** (bool *modifiable*) –

  If True, other users can see if you deleted them from your contact book

  - The users can see it only if they is_premium users, or by using meapi ;)
  - Must be enabled in order to use *who_deleted()*.

- **who_watched_enabled** (bool *modifiable*) –

  If True, other users can see if you watch their profile.

  - The users can see it only if they `is_premium` users, or by using `meapi` ;)
  - Must be enabled in order to use *who_watched()*.

- **comments_enabled** (`bool` *modifiable*) –

  Allow other users to *publish_comment()* on your profile.

  - You always need to *approve_comment()* before they are published.
  - You can block spesfic users from commenting on your profile with *block_comments()*.

- **location_enabled** (`bool` *modifiable*) – Allow other users ask to see your location.
- **mutual_contacts_available** (`bool` *modifiable*) –

  If `True`, other users can see your mutual contacts.

  - See *friendship()* for more information.

- **notifications_enabled** (`bool` *modifiable*) –

  Get notify on new messages.

  - See *get_notifications()* for more information.

- **who_deleted_notification_enabled** (`bool` *modifiable*) –

  Get notify on who deleted you from your contact book.

  - You will only receive notifications if `who_deleted_enabled` is `True`.

- **who_watched_notification_enabled** (`bool` *modifiable*) –

  Get notify on who watched your profile.

  - You will only receive notifications if `who_watched_enabled` is `True`.

- **comments_notification_enabled** (`bool` *modifiable*) –

  Get notify on new comments, likes etc.

  - You will only receive notifications if `comments_enabled` is `True`.

- **birthday_notification_enabled** (`bool` *modifiable*) – Get notify on contact birthday.
- **distance_notification_enabled** (`bool` *modifiable*) – Get notify on contacts distance.
- **names_notification_enabled** (`bool` *modifiable*) – Get notify when someone saved you in is contacts book, new joined contacts to Me, new rename approve and more.
- **system_notification_enabled** (`bool` *modifiable*) – Get notify on system messages: spam reports, mutual requests and more.
- **contact_suspended** (`bool`) – If *True*, the contact is suspended.
- **language** (`str` *modifiable*) – Language of the notifications.
- **last_backup_at** (`datetime` *optional*) – Last backup time.
- **last_restore_at** (`datetime` *optional*) – Last restore time.
- **spammers_count** (`int`) – Number of spammers.

BlockedNumber model

class meapi.models.blocked_number.**BlockedNumber**

Represents a blocked number.

- For more information about blocked numbers

Example

```
>>> from meapi import Me
>>> blocked_numbers = Me.get_blocked_numbers()
>>> for blocked_number in blocked_numbers: blocked_number.unblock()
```

Parameters

- **block_contact** (bool) – This contact cannot call or text you.
- **me_full_block** (bool) – This contact cannot watch your Me profile (And neither will you be able to view his).
- **phone_number** (int) – The phone number of the contact.

MutualContact model

class meapi.models.mutual_contact.**MutualContact**

Represents a Mutual contact between you and another user.

Parameters

- **name** (str) – The user's fullname.
- **phone_number** (int) – The user's phone number.
- **date_of_birth** (date) – The user's date of birth.
- **uuid** (str *optional*) – The user's unique ID.

Get the contact as Vcard

as_vcard().

Block this contact

block().

Unblock this contact

unblock().

Report this contact as spam

report_spam().

Others

class meapi.models.others.**NewAccountDetails**

Account details for new account registration.

Parameters

- **first_name** (str) – First name to use.
- **last_name** (str | None) – Last name to use. *Default:* None.
- **email** (str | None) – Email to use. *Default:* None.

class meapi.models.others.**Call**

class meapi.models.others.**Contact**

class meapi.models.others.**CallType**

Call type enum.

class meapi.models.others.**RequestType**

Request type enum.

7.2.4 Credentials Manager

The credentials managers allows you to store your credentials in your own way.

meapi, needs to store your credentials (access token, refresh token etc.) in order to be able to use them later on without the need to login again every time you want to use the API.

There are number of credentials managers that are already implemented in the project:

- *JsonCredentialsManager*: **stores the credentials in a json file (meapi_credentials.json by default).**
  - This is the default credentials manager.
- *MemoryCredentialsManager*: **stores the credentials in memory.**
  - The credentials will be lost when the program exits

And more… (see the list below)

How to use a credentials manager? simple as that:

```
>>> from me import Me
>>> me = Me(phone_number=972123456789, credentials_manager=YourCredentialsManager())
```

- You are more than welcome to create your own credentials manager and open a pull request to add it to the project.

Creating your own custom CredentialsManager

- You must implement the methods `get`, `set`, `update` and `delete` in order to allow `meapi` to store and manage the credentials.

class meapi.credentials_managers.**CredentialsManager**

Abstract class for credentials managers.

- You can implement your own credentials manager to store credentials in your own way.

abstract delete(*phone_number: str*)

Delete the credentials by phone_number key.

- If the credentials are not in the manager, do nothing.
- You can implement your own idea of what `delete` means (e.g. change the state of your credentials).

abstract get(*phone_number: str*) → Optional[Dict[str, str]]

Get the credentials by phone_number key.

- If the credentials are not in the manager, return `None`.
- The keys of the dict must be: `access`, `refresh` and `pwd_token`, see example below.

Parameters
phone_number (`str`) – The phone number of the client.

Returns
Optional dict with the credentials. see example below.

Return type
dict

Example for return value:

```
{
    'access': 'xxx',
    'refresh': 'xxx',
    'pwd_token': 'xxx'
}
```

abstract set(*phone_number: str*, *data: Dict[str, str]*)

Set the credentials by phone_number key.

- If the credentials are already in the manager, update them.

Parameters

- **phone_number** (*str*) – The phone number of the client.
- **data** (*dict*) – Dict with credentials. see example below.

Example for `data`:

```
{
    'access': 'xxx',
    'refresh': 'xxx',
    'pwd_token': 'xxx',
}
```

abstract update(*phone_number: str*, *access_token: str*)

Update the access token in the credentials by `phone_number` key.

Parameters

- **phone_number** (*str*) – The phone number of the client.
- **access_token** (*str*) – The new access token.

Example: Using a database

Let's say you want to store the credentials in a database

- In this example we will use the Pony ORM , but you can use any other ORM or database library you want.

Creating the database

First, we need to create the database and the table that will store the credentials:

```
from pony.orm import Database, PrimaryKey, Required

db = Database()
db.bind(provider='sqlite', filename='meapi_credentials.sqlite', create_db=True)

class MeUser(db.Entity):
    _table_ = 'me_user'
    phone_number = PrimaryKey(str)
    pwd_token = Required(str)
    access = Required(str)
    refresh = Required(str)
    status = Required(str, default='active')

db.generate_mapping(create_tables=True)
```

Implementing the CredentialsManager interface

Create a new class that implements the *CredentialsManager* interface:

```
from meapi.credentials_manager import CredentialsManager
from typing import Optional, Dict
from pony.orm import db_session, TransactionIntegrityError

class DatabaseCredentialsManager(CredentialsManager):
    @db_session
```

(continues on next page)

(continued from previous page)

```
    def set(self, phone_number: str, data: dict):
        try:
            User(phone_number=phone_number, **data)
        except TransactionIntegrityError: # phone_number already exists
            User[phone_number].set(status='active', **data)

    @db_session
    def get(self, phone_number: str) -> Optional[Dict[str, str]]:
        user = User.get(phone_number=phone_number)
        return user.to_dict(only=['pwd_token', 'access', 'refresh']) if
            (user and user.status == 'active') else None

    @db_session
    def update(self, phone_number: str, access_token: str):
        User[phone_number].access = access_token

    @db_session
    def delete(self, phone_number: str):
        User[phone_number].status = 'inactive' # We don't want actually to delete the␣
↪user from the database
```

Using the CredentialsManager

Now we can use the credentials manager by passing it to the *Me* class:

```
>>> from meapi import Me
>>> dbcm = DatabaseCredentialsManager()
>>> me = Me(phone_number=972123456789, activation_code='123456', credentials_
↪manager=dbcm)
```

Available Credentials Managers

class meapi.credentials_managers.json_files.**JsonCredentialsManager**

Json Credentials Manager

- This class is used to store the credentials in a json file/s.

Parameters

- **config_file** (-) – (str) The config json file path. *Default:* `meapi_credentials.json`.

```
{
    "123456789": {"pwd_token": "xxx", "access": "xxx", "refresh":
↪"xxx"},
    "987654321": {"pwd_token": "xxx", "access": "xxx", "refresh":
↪"xxx"},
    "123456789": {"pwd_token": "xxx", "access": "xxx", "refresh":
```

(continues on next page)

(continued from previous page)

```
↪"xxx"}
}
```

- **separate_files** (-) – (bool) If True, each phone number will have its own file: credentials_dir/phone_number.json. *Default:* False.
- **directory** (-) – (str) The directory where the credentials files will be stored. *Default:* meapi_credentials.

```
.
├── meapi_credentials
│   ├── 123456789.json -> {"pwd_token": "xxx", "access": "xxx",
↪"refresh": "xxx"}
│   ├── 987654321.json -> {"pwd_token": "xxx", "access": "xxx",
↪"refresh": "xxx"}
│   └── 123456789.json -> {"pwd_token": "xxx", "access": "xxx",
↪"refresh": "xxx"}
└──────────────────────
```

Raises

- **FileExistsError** – If the config file is not a valid json file.

Example

```
>>> from meapi.credentials_managers.json_files import JsonCredentialsManager
>>> from meapi import Me
>>> jcm = JsonCredentialsManager(separate_files=True, directory='/home/david/
↪credentials')
>>> me = Me(phone_number='123456789', credentials_manager=jcm)
```

class meapi.credentials_managers.memory.**MemoryCredentialsManager**

Memory Credentials Manager.

- This class is storing the credentials in memory.
- The data will be lost when the program ends.

class meapi.credentials_managers.redis.**RedisCredentialsManager**

Redis Credentials Manager.

- This class is used to store the credentials in a redis cache.

Parameters

redis (-) – (redis.Redis) The redis object of redis-py library. (https://github.com/redis/redis-py)

Examples

```
>>> from meapi import Me
>>> from redis import Redis
>>> from meapi.credentials_managers.redis import RedisCredentialsManager
>>> redis = Redis(host='localhost', port=6379, db=0)
>>> rcm = RedisCredentialsManager(redis)
>>> me = Me(phone_number=972123456789, credentials_manager=rcm)
```

class meapi.credentials_managers.flask.**FlaskSessionCredentialsManager**

Flask Session Credentials Manager

- This class is used to store the credentials in a flask session.

Parameters

session (-) – (flask.sessions.Session) The flask session.

class meapi.credentials_managers.django.**DjangoSessionCredentialsManager**

Django Session Credentials Manager

- This class is used to store the credentials in a django session.

Parameters

session (-) – (django.contrib.sessions.backends.base.SessionBase) The django session.

7.2.5 Exceptions

This is a list of all exceptions that may be raised by the library.

On the top level, there are two exceptions:

- *MeApiException*: Raised when the API returns an error.
- *MeException*: Raised when the library itself has an error.

It is recommended to catch *MeApiException* and *MeException* separately. Here is an example:

```
from meapi import Me
from meapi.utils.exceptions import *

# Getting some user input:
phone_number = input("Enter your phone number: ")
activation_code = input("Enter your activation code: ")

try:
    me = Me(phone_number=phone_number, activation_code=activation_code)
except MeApiException as e:
    if isinstance(e, IncorrectActivationCode):
        print("Incorrect activation code!", e.reason)
    elif isinstance(e, ActivationCodeExpired):
        print("Activation code has expired!", e.reason)
    else:  # There are more exceptions, but they are not listed here (See the doc of the␣
↪Me class)
```

(continues on next page)

(continued from previous page)

```
        print("Unknown error!", e.msg, e.reason)
except MeException as e:
    if isinstance(e, NotValidPhoneNumber):
        print("Not a valid phone number!")
    else:
        print("Unknown error!", e.msg)
except ValueError as e:
    print("Wrong activation code. It must be a 6-digit number.")

# If there is no exception, the code will continue here.
```

This are the exceptions that may be raised by the API:

class meapi.utils.exceptions.**MeApiException**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception if http status code is bigger than 400.

- Base class for all api exceptions.

Parameters

- **http_status** (*int*) – status code of the http request. =>400.
- **msg** (*str*) – api error msg. for example: api_incorrect_activation_code.
- **reason** (*str*) – Human reason to the error.

class meapi.utils.exceptions.**IncorrectPwdToken**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the pwd token is incorrect. Happens when opening the account elsewhere.

class meapi.utils.exceptions.**NewAccountException**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the account is new. Happens when trying to login to a new account. Provide NewAccountDetails in the Me constructor to continue.

class meapi.utils.exceptions.**UnfinishedRegistration**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the registration is unfinished. Happens when trying to login to an account that is not registered. Provide NewAccountDetails in the Me constructor to continue.

class meapi.utils.exceptions.**PhoneNumberDoesntExists**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the phone number doesn't exist in me database.

class meapi.utils.exceptions.**IncorrectActivationCode**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the activation code is incorrect.

class meapi.utils.exceptions.**MaxValidateReached**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the max validate attempts via WhatsApp or Telegram reached.

class meapi.utils.exceptions.**BlockedMaxVerifyReached**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the max verify attempts via sms or call reached.

class meapi.utils.exceptions.**ActivationCodeExpired**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the activation code is correct but expired.

class meapi.utils.exceptions.**SearchPassedLimit**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the search passed the limit. Happens when searching for a phone number too many times. The limit in the unofficial authentication method is 350 searches per day.

class meapi.utils.exceptions.**ProfileViewPassedLimit**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the profile view passed the limit. Happens when viewing profiles too many times. The limit in the unofficial authentication method is 500 views per day.

class meapi.utils.exceptions.**UserCommentsDisabled**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the user comments are disabled. Happens when trying to publish a comment to a user that disabled comments.

class meapi.utils.exceptions.**UserCommentsPostingIsNotAllowed**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the user comments posting is not allowed. Happens when trying to publish a comment to a user that blocked you from commenting.

class meapi.utils.exceptions.**CommentAlreadyApproved**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the comment is already approved. Happens when trying to approve a comment that is already approved.

class meapi.utils.exceptions.**CommentAlreadyIgnored**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the comment is already ignored. Happens when trying to ignore a comment that is already ignored.

class meapi.utils.exceptions.**BlockedAccount**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the account is blocked. Happens when trying to login to a blocked account.

class meapi.utils.exceptions.**ForbiddenRequest**(*http_status: int*, *msg: str*, *reason: Optional[str] = None*)

Raise this exception when the request is forbidden.

- Happens in official authentication method when the access token is expired.

This are the exceptions that may be raised by the library:

class meapi.utils.exceptions.**MeException**(*msg: str*)

Raises this exception when something goes wrong.

- Base class for all library exceptions.

Parameters

msg (*str*) – Reason of the exception.

class meapi.utils.exceptions.**NotValidPhoneNumber**(*msg: Optional[str] = None*)

Raise this exception when the phone number is not valid.

class meapi.utils.exceptions.**NotValidAccessToken**(*msg: Optional[str] = None*)

Raise this exception when the access token is not valid.

class meapi.utils.exceptions.**NotLoggedIn**(*msg: Optional[str] = None*)

Raise this exception when the user is not logged in.

- Happens when trying to call a method after calling `me_client.logout(args)`

Parameters

msg (*str*) – Reason of the exception.

class meapi.utils.exceptions.**NeedActivationCode**(*msg: Optional[str] = None*)

Raise this exception when the activation code is needed.

- Happens when trying to login to a new account without providing the activation code in the constructor and the client initialized with `interaction_mode = False`.

Parameters

msg (*str*) – Reason of the exception.

class meapi.utils.exceptions.**ContactHasNoUser**(*msg: Optional[str] = None*)

Raise this exception when the contact has no user.

class meapi.utils.exceptions.**FrozenInstance**(*cls: Type[MeModel], attr: str, msg: Optional[str] = None*)

Raise this exception when trying to change a frozen instance.

- In some cases, the library uses frozen instances to prevent changing the attributes because in some models, reassigning the attributes will actually change the data on the server.

Parameters

- **cls** (*MeModel*) – The class of the frozen instance.
- **attr** (*str*) – The attribute that is trying to be changed.
- **msg** (*str*) – Reason of the exception. override the default message.

class meapi.utils.exceptions.**BrokenCredentialsManager**(*msg: Optional[str] = None*)

Raise this exception when the credentials manager is broken.

- Happens when login was successful but the credentials manager not providing the access token.

Parameters

msg (*str*) – Reason of the exception.

7.3 Examples

IN PROGRESS

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

M

N

P

R

S

U

W