vk_maria

lxstvayne

февр. 13, 2023

Оглавление

1 Официальные ресурсы vk_maria **3**

2 Сильные стороны **5**

3 Contents **7**
3.1 Установка . . . 7
3.2 Быстрый старт . . . 8
3.3 Обработка событий . . . 9
3.4 Обработчики событий . . . 10
3.5 Загрузка файлов . . . 11
3.6 Клавиатуры . . . 11
3.7 Конечные автоматы (FSM) . . . 12
3.8 API . . . 14

4 Indices and tables **271**

Содержание модулей Python **273**

Алфавитный указатель **275**

vk_maria очень простой фреймворк для создания ботов сообществ Vk, написанный на Python 3.8.

Глава 1

Официальные ресурсы vk_maria

- Новости: @vk_maria
- Чат комьюнити: @vk_maria_ru
- Pip: vk_maria
- Docs: ReadTheDocs
- Source: Github репозиторий
- Issues/Bug tracker: Github issues tracker

Глава 2

Сильные стороны

- Простота и удобство
- Наличие конечных автоматов (FSM)
- Типизированная

Глава 3

Contents

3.1 Установка

3.1.1 Установка с помощью pip:

```
(.venv) $ pip install vk_maria
```

3.1.2 Установка с github:

```
(.venv) $ git clone https://github.com/lxstvayne/vk_maria
(.venv) $ cd vk_maria
(.venv) $ python setup.py install
```

Обычно рекомендуется использовать первый способ.

Хотя библиотека и готова к использованию, она всё ещё находится на стадии разработки, поэтому не забывайте регулярно её обновлять:

```
(.venv) $ pip install vk_maria --upgrade
```

3.2 Быстрый старт

Прежде всего вы должны получить ключ доступа вашего сообщества.

3.2.1 Простейший пример

Класс `Vk` инкапсулирует все методы работы с токеном сообщества.

Создайте файл **echo_bot.py**. Откройте его и создайте экземпляр класса `Vk`:

```
from vk_maria import Vk, types
from vk_maria.dispatcher import Dispatcher


vk = Vk(access_token='token')
```

Примечание: Обязательно замените token ключом доступа вашего сообщества.

Затем создайте экземпляр класса `Dispatcher` передав ему в качестве аргумента `vk`:

```
dp = Dispatcher(vk)
```

После этого нам необходимо зарегистрировать обработчик событий. Обработчики событий определяют фильтры, которые должно пройти событие. Если событие проходит фильтры, вызывается декорированная функция и входящее событие передаётся в качестве аргумента.

Давайте определим обработчик событий, который будет обрабатывать все входящие сообщения от пользователя в личные сообщения сообщества и отвечать на команду Начать:

```
@dp.message_handler(text='Начать')
def send_welcome(event: types.Message):
    vk.messages_send(user_id=event.message.from_id, message='Добро пожаловать!')
```

Добавим ещё один обработчик:

```
@dp.message_handler()
def echo(event: types.Message):
    event.answer(event.message.text)
```

Как вы могли заметить, `event.answer` является удобным аналогом `vk.messages_send`.

Декорированная функция может иметь произвольное имя, однако она должна принимать минимум 1 параметр (event).

Примечание: Все обработчики тестируются в том порядке, в котором они были объявлены.

Отлично, теперь у нас есть простой бот, который отвечает на сообщение Начать приветствием и повторяет остальные отправленные сообщения. Чтобы запустить бота добавьте в исходный код следующее:

```
dp.start_polling(debug=True)
```

Примечание: Параметр debug отвечает за вывод в консоль всех происходящих событий

Вот и всё! Наш исходный файл теперь выглядит так:

```
from vk_maria import Vk, types
from vk_maria.dispatcher import Dispatcher


vk = Vk(access_token='token')
dp = Dispatcher(vk)


@dp.message_handler(text='Начать')
def send_welcome(event: types.Message):
    vk.messages_send(user_id=event.message.from_id, message='Добро пожаловать!')


@dp.message_handler()
def echo(event: types.Message):
    event.answer(event.message.text)


if __name__ == '__main__':
    dp.start_polling(debug=True)
```

Чтобы запустить бота, просто откройте терминал, введите `python echo_bot.py` и протестируйте его.

3.3 Обработка событий

3.3.1 Лонгполлинг

События можно обрабатывать двумя способами

С помощью цикла

```
from vk_maria import Vk
from vk_maria.longpoll import LongPoll

vk = Vk(access_token='token')
lp = LongPoll(vk)

for event in lp.listen():
    ...
```

С помощью диспетчера событий

```
from vk_maria import Vk
from vk_maria.dispatcher import Dispatcher

vk = Vk(access_token='token')
dp = Dispatcher(vk)

@dp.message_handler()
def handler(event: types.Message)
    ...

if __name__ == '__main__':
    dp.start_polling()
```

3.4 Обработчики событий

Обработчик событий это функция с декоратором `event_handler()`. Он определяет фильтры для обрабатываемых событий.

```
@dp.event_handler(event_type, *filters, **bound_filters)
def handler(event):
    ...
```

Для удобной работы с сообщениями, существует декоратор `message_handler(*filters, **bound_filters)`, в который по стандарту передаётся `types.EventType.MESSAGE_NEW`.

Связанные фильтры устанавливаются следующим образом: `name=argument`

Таблица 1: Доступные связанные фильтры

Название	Аргументы	Условие
event_type	types.EventType	**True**, если типы событий совпадают.
text	Строка	**True**, если текст сообщения совпадает с аргументом **text**
regexp	Регулярное выражение или подстрока	**True**, если подстрока находится в сообщении или строка проходит проверку на наличие шаблона регулярного выражения (Подробнее Python Regular Expressions).
commands	Список строк	**True**, если текст сообщения совпадает с одной из команд
frm	От кого обрабатывать события (user, chat, group) по умолчанию user	**True**, если поле **from_(user, chat, group)** соответственно равно аргументу frm
state	Состояние автомата	**True**, если текущее состояние равно состаянию аргумента state

Чтобы начать обрабатывать события, необходимо запустить `start_polling()`. Для удобства разработки можно передать параметр `debug=True`. Тогда все происходящие события будут красиво выводиться в консоль.

3.4.1 Собственные фильтры

Чтобы определить собственный фильтр, необходимо написать класс и переопределить функцию `check`.

```
from vk_maria.dispatcher.filters import AbstractFilter

class AdminFilter(AbstractFilter):
    def check(self, event: types.Message):
        return event.message.peer_id == 1234567890
```

И передать его в обработчик события:

```
@dp.message_handler(AdminFilter, commands=['/start'])
def cmd_start(event: types.Message):
    event.reply("Hi there! What's your name?")
```

Количество пользовательских фильтров неограничено, их необходимо передавать первым аргументом через запятую.

3.5 Загрузка файлов

Класс `Upload` реализует готовые функции для загрузки файлов на сервера Вконтакте.

Доступные методы:

- `photo(photo)`
- `set_chat_photo(file, chat_id, **kwargs)`
- `set_group_cover_photo(photo)`
- `document(document, peer_id, **kwargs)`

Параметры **photo** и **document** могут быть как строкой относительного пути к файлу, так и файлом открытым с помощью **open()** на бинарное чтение **rb** стандартной библиотеки **Python**.

3.6 Клавиатуры

Клавиатуры можно создавать двумя способами

3.6.1 KeyboardModel

С помощью определения класса

```
from vk_maria.types import KeyboardModel, Button, Color

class TestKeyboard(KeyboardModel):

    one_time = True

    row1 = [
        Button.Text(Color.PRIMARY, 'Кнопка 1'),
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        Button.Text(Color.PRIMARY, 'Кнопка 2')
    ]
    row2 = [
        Button.Text(Color.PRIMARY, 'Кнопка 3'),
        Button.Text(Color.PRIMARY, 'Кнопка 4')
    ]
```

3.6.2 KeyboardMarkup

С помощью генерации через код

```
from vk_maria.types import KeyboardMarkup, Button, Color

markup = KeyboardMarkup(one_time=True)
markup.add_button(Button.Text(Color.PRIMARY, 'Кнопка 1'))
markup.add_button(Button.Text(Color.PRIMARY, 'Кнопка 2'))
markup.add_row()
markup.add_button(Button.Text(Color.PRIMARY, 'Кнопка 3'))
markup.add_button(Button.Text(Color.PRIMARY, 'Кнопка 4'))
```

3.7 Конечные автоматы (FSM)

Если вашему боту необходима система диалогов, то в дело вступают конечные автоматы. Это некая математическая модель, которая представляет собой набор состояний, которые переключаются в определённых условиях. В библиотеке это реализовано с помощью классов

```
from vk_maria.dispatcher.fsm import StatesGroup, State, MemoryStorage, FSMContext

class Form(StatesGroup):
    waiting_for_name: State
    waiting_for_age: State
    waiting_for_gender: State
```

Необходимо описать в классе переменные состояний.

Чтобы имелась возможность запоминать текущие состояния для диалогов всех пользователей, в класс `Dispatcher` необходимо передать экземпляр хранилища состояний.

```
dp = Dispatcher(vk, MemoryStorage())
```

В библиотеке на данный момент реализованы следующие хранилища состояний: `MemoryStorage`, `JSONStorage`, `PickleStorage`.

3.7.1 Пример реализации

```
from vk_maria import Vk, types
from vk_maria.dispatcher import Dispatcher
from vk_maria.dispatcher.fsm import StatesGroup, State, MemoryStorage, FSMContext
from vk_maria.types import KeyboardMarkup, Button, Color, RemoveReplyMarkup


vk = Vk(access_token='token')
dp = Dispatcher(vk, MemoryStorage())

GENDERS = ('Male', 'Female', 'Other')


class Form(StatesGroup):
    waiting_for_name: State
    waiting_for_age: State
    waiting_for_gender: State


@dp.message_handler(commands=['/start'])
def cmd_start(event: types.Message):
    event.reply("Hi there! What's your name?")
    Form.waiting_for_name.set()


@dp.message_handler(state=Form.waiting_for_name)
def process_name(event: types.Message, state: FSMContext):
    state.update_data(name=event.message.text)
    event.reply("How old are you?")
    Form.next()


@dp.message_handler(state=Form.waiting_for_age)
def process_age(event: types.Message, state: FSMContext):
    state.update_data(age=event.message.text)

    markup = KeyboardMarkup(one_time=False)
    for gender in GENDERS:
        markup.add_button(Button.Text(Color.PRIMARY, gender))

    event.reply('What is your gender?', keyboard=markup)
    Form.next()


@dp.message_handler(state=Form.waiting_for_gender)
def process_gender(event: types.Message, state: FSMContext):
    if event.message.text not in GENDERS:
        return event.reply('Bad gender name. Choose your gender from the keyboard.')

    state.update_data(gender=event.message.text)
    user_data = state.get_data()
    event.answer(f'Hi! Nice to meet you, {user_data["name"]}\n'
```

(continues on next page)

(продолжение с предыдущей страницы)

```
                f'Age: {user_data["age"]}\n'
                f'Gender: {user_data["gender"]}', keyboard=RemoveReplyMarkup)
    Form.finish()


if __name__ == '__main__':
    dp.start_polling(debug=True)
```

Примечание: Чтобы каждый раз не писать FSMContext.get_current(), можно передать аргумент state: FSMContext в функцию-обработчик.

Чтобы отлавливать состояние, передаётся аргумент `state` для `message_handler`

3.8 API

vk_maria

3.8.1 vk_maria

vk_maria.api

vk_maria.dispatcher

vk_maria.exceptions

vk_maria.longpoll

vk_maria.mixins

vk_maria.responses

vk_maria.types

vk_maria.upload

vk_maria.utils

vk_maria.vk_types

vk_maria.api

Classes

ApiMethod	**param access_token**
Vk	Читайте подробнее про методы https://vk.com/dev/methods

vk_maria.api.ApiMethod

class ApiMethod(*access_token*, *api_version*)

Базовые классы: object

Параметры

- access_token (str) –
- api_version (str) –

Methods

Attributes

base_url
http
last_request
rps_delay

base_url: str = 'https://api.vk.com/method/'

http = <requests.sessions.Session object>

rps_delay = 0.05

last_request = 0.0

__call__(**args*, ***kwargs*)

Call self as a function.

vk_maria.api.Vk

class Vk(*access_token*, *api_version='5.126'*)

Базовые классы: object

Читайте подробнее про методы https://vk.com/dev/methods

Параметры

- access_token (str) – Токен сообщества
- api_version (str, default: '5.126') – Версия Api

Methods

board_delete_comment	**type topic_id** int
board_restore_comment	**type topic_id** int
docs_get_messages_upload_server	**type peer_id** int
docs_get_wall_upload_server	
docs_save	**type file**
docs_search	**type q** str
groups_add_address	**type title** str
groups_delete_address	**type address_id** int
groups_disable_online	
groups_edit	**type title** Optional[str], default: None

continues on next page

Таблица 2 – продолжение с предыдущей страницы

groups_edit_address	**type address_id** int
groups_enable_online	
groups_get_banned	**type fields** Optional[List[str]], default: None
groups_get_by_id	**type fields** Optional[List[str]], default: None
groups_get_longpoll_server	
groups_get_members	**type sort** str, default: 'id_asc'
groups_get_online_status	
groups_get_token_permissions	
groups_is_member	**type user_id** Optional[int], default: None
groups_set_settings	**type messages** Optional[int], default: None
market_edit_order	**type user_id** int
market_get_group_orders	**type offset** Optional[int], default: None
market_get_order_by_id	**type order_id** int
market_get_order_items	**type order_id** int

continues on next page

Таблица 2 – продолжение с предыдущей страницы

messages_create_chat	**type user_ids** List[int]
messages_delete	**type message_ids** Optional[List[int]], default: None
messages_delete_chat_photo	**type chat_id** int
messages_delete_conversation	**type user_id** Optional[int], default: None
messages_edit	**type peer_id** int
messages_edit_chat	**type chat_id** int
messages_get_by_conversation_message_id	**type peer_id** int
messages_get_by_id	**type message_ids** List[int]
messages_get_conversation_members	**type peer_id** int
messages_get_conversations	**type offset** int, default: 0
messages_get_conversations_by_id	**type peer_ids** List[int]
messages_get_history	**type offset** Optional[int], default: None

continues on next page

Таблица 2 – продолжение с предыдущей страницы

messages_get_history_attachments	**type peer_id** int
messages_get_important_messages	**type count** int, default: 20
messages_get_intent_users	**type intent** str
messages_get_invite_link	**type peer_id** int
messages_get_longpoll_history	**type ts** int
messages_get_longpoll_server	**type need_pts** Optional[int], default: None
messages_is_messages_from_group_allowed	**type user_id** int
messages_mark_as_answered_conversation	**type peer_id** int
messages_mark_as_important_conversation	**type peer_id** int
messages_mark_as_read	**type message_ids** Optional[List[int]], default: None
messages_pin	**type peer_id** int
messages_remove_chat_user	**type chat_id** int

continues on next page

Таблица 2 – продолжение с предыдущей страницы

messages_restore	**type message_id** int
messages_search	**type q** Optional[str], default: None
messages_search_conversations	**type q** str
messages_send	**type user_id** Optional[int], default: None
messages_send_message_event_answer	**type event_id** str
messages_set_activity	**type user_id** int
messages_set_chat_photo	**type file** str
messages_unpin	**type peer_id** int
photos_get_chat_upload_server	**type chat_id** int
photos_get_messages_upload_server	
photos_get_owner_cover_photo_upload_serve	**type crop_x** Optional[float], default: None
photos_save_messages_photo	**type photo** str
photos_save_owner_cover_photo	**type hash** str

continues on next page

Таблица 2 – продолжение с предыдущей страницы

podcasts_search_podcast	**type search_string** str
storage_get	**type key** Optional[str], default: None
storage_get_keys	**type user_id** int
storage_set	**type key** str
stories_delete	**type owner_id** int
stories_get	**type owner_id** int
stories_get_by_id	**type stories** List[int]
stories_get_photo_upload_server	**type add_to_news** Optional[int], default: None
stories_get_replies	**type owner_id** int
stories_get_stats	**type owner_id** int
stories_get_video_upload_server	**type user_ids** Optional[List[int]], default: None
stories_get_viewers	**type owner_id** int

continues on next page

Таблица 2 – продолжение с предыдущей страницы

stories_hide_all_replies	**type owner_id** int
stories_hide_reply	**type owner_id** int
stories_save	**type upload_results** List[str]
users_get	**type user_ids** List[int]
utils_check_link	**type url** str
utils_get_link_stats	**type key** str
utils_get_server_time	Возвращает число, соответствующее времени в UnixTime.
utils_get_short_link	**type url** str
utils_resolve_screen_name	**type screen_name** str
wall_close_comments	**param owner_id**
wall_create_comment	**type post_id** int

messages_create_chat(*user_ids*, *title*)

Параметры

- **user_ids** (List[int]) – Идентификаторы пользователей, которых нужно включить в мультидиалог.
- **title** (str) – Название беседы.

messages_delete(*message_ids=None*, *peer_id=None*, *spam=None*, *delete_for_all=None*, *cmids=None*)

Параметры

- `message_ids` (`Optional[List[int]]`, default: `None`) – Список идентификаторов сообщений.
- `spam` (`Optional[int]`, default: `None`) – Помечает сообщения как спам.
- `delete_for_all` (`Optional[int]`, default: `None`) – Удаление для всех.
- `cmids` (`Optional[List[int]]`, default: `None`) – Conversation Message Ids
- `peer_id` (`Optional[int]`, default: `None`) –

`messages_delete_chat_photo(`*chat_id*`)`

Параметры

`chat_id` (`int`) – Идентификатор беседы.

`messages_delete_conversation(`*user_id=None*, *peer_id=None*`)`

Параметры

- `user_id` (`Optional[int]`, default: `None`) – Идентификатор пользователя. Если требуется очистить историю беседы, используйте peer_id.
- `peer_id` (`Optional[int]`, default: `None`) – Идентификатор назначения.

`messages_edit(`*peer_id*, *message=None*, *lat=None*, *long=None*, *attachment=None*, *keep_forward_messages=None*, *keep_snippets=None*, *dont_parse_links=None*, *message_id=None*, *conversation_message_id=None*, *template=None*, *keyboard=None*`)`

Параметры

- `peer_id` (`int`) – Идентификатор назначения.
- `message` (`Optional[str]`, default: `None`) – Текст сообщения. Обязательный параметр, если не задан параметр attachment.
- `lat` (`Optional[float]`, default: `None`) – Географическая широта (от -90 до 90).
- `long` (`Optional[float]`, default: `None`) – Географическая долгота (от -180 до 180).
- `attachment` (default: `None`) – Медиавложения к личному сообщению, перечисленные через запятую.
- `keep_forward_messages` (`Optional[int]`, default: `None`) – Сохранить прикреплённые пересланные сообщения.
- `keep_snippets` (`Optional[int]`, default: `None`) – Сохранить прикреплённые внешние ссылки (сниппеты).
- `dont_parse_links` (`Optional[int]`, default: `None`) – Не создавать сниппет ссылки из сообщения.
- `message_id` (`Optional[int]`, default: `None`) – Идентификатор сообщения.
- `conversation_message_id` (`Optional[int]`, default: `None`) – Идентификатор сообщения в беседе.
- `template` (default: `None`) – Объект, описывающий шаблоны сообщений.
- `keyboard` (default: `None`) – Объект, описывающий клавиатуру бота.

messages_edit_chat(*chat_id*, *title*)

Параметры

- chat_id (int) – Идентификатор беседы.
- title (str) – Новое название для беседы.

messages_get_by_conversation_message_id(*peer_id*, *conversation_message_ids*, *extended=None*, *fields=None*)

Параметры

- peer_id (int) – Идентификатор назначения.
- conversation_message_ids (List[int]) – Идентификаторы сообщений. Максимум 100 идентификаторов.
- extended (Optional[int], default: None) – Возвращать дополнительные поля.
- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.

messages_get_by_id(*message_ids*, *preview_length=None*, *extended=None*, *fields=None*)

Параметры

- message_ids (List[int]) – Идентификаторы сообщений. Максимум 100 идентификаторов.
- preview_length (Optional[int], default: None) – Количество символов, по которому нужно обрезать сообщение.
- extended (Optional[int], default: None) – Возвращать дополнительные поля.
- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.

messages_get_conversation_members(*peer_id*, *fields*)

Параметры

- peer_id (int) – Идентификатор назначения.
- fields (List[str]) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.

messages_get_conversations(*offset=0*, *count=20*, *filter='all'*, *extended=None*, *start_message_id=None*, *fields=None*)

Параметры

- offset (int, default: 0) – Смещение, необходимое для выборки определенного подмножества результатов.
- count (int, default: 20) – Максимальное число результатов, которые нужно получить.
- filter (str, default: 'all') – Фильтр. Возможные значения: all, unread, important, unanswered.
- extended (Optional[int], default: None) – Возвращать дополнительные поля.
- start_message_id (Optional[int], default: None) – Идентификатор сообщения, начиная с которого нужно возвращать беседы.

- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.

messages_get_conversations_by_id(*peer_ids*, *extended=None*, *fields=None*)

Параметры

- peer_ids (List[int]) – Идентификаторы назначений.
- extended (Optional[int], default: None) – Возвращать дополнительные поля.
- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.

messages_get_history(*offset=None*, *count=20*, *user_id=None*, *peer_id=None*, *start_message_id=None*, *rev=None*, *extended=None*, *fields=None*)

Параметры

- offset (Optional[int], default: None) – Смещение, необходимое для выборки определенного подмножества сообщений
- count (int, default: 20) – Количество сообщений, которое необходимо получить (но не более 200)
- user_id (Optional[int], default: None) – Идентификатор пользователя, историю переписки с которым необходимо вернуть.
- peer_id (Optional[int], default: None) – Идентификатор назначения.
- start_message_id (Optional[int], default: None) – Идентификатор сообщения, начиная с которого нужно возвращать беседы.
- rev (Optional[int], default: None) – Возвращать сообщения в хронологическом порядке (по умолчанию).
- extended (Optional[int], default: None) – Возвращать дополнительные поля.
- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.

messages_get_history_attachments(*peer_id*, *media_type=None*, *start_from=None*, *count=20*, *photo_sizes=None*, *fields=None*, *preserve_order=None*, *max_forwards_level=45*)

Параметры

- peer_id (int) – Идентификатор назначения.
- media_type (Optional[List[str]], default: None) – Тип материалов, который необходимо вернуть.
- start_from (*str / None*) –
- count (*int*) –
- photo_sizes (*int / None*) –
- fields (*List[str] / None*) –
- preserve_order (*int / None*) –
- max_forwards_level (*int*) –

Доступные значения: photo, video, audio, doc, link, market, wall, share.

Параметры

- start_from (Optional[str], default: None) – Смещение, необходимое для выборки определенного подмножества объектов.
- count (int, default: 20) – Количество объектов, которое необходимо получить (но не более 200).
- photo_sizes (Optional[int], default: None) – Параметр, указывающий нужно ли возвращать ли доступные размеры фотографии в специальном формате.
- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.
- preserve_order (Optional[int], default: None) – Параметр, указывающий нужно ли возвращать вложения в оригинальном порядке.
- max_forwards_level (int, default: 45) – Максимальная глубина вложенности пересланных сообщений.
- peer_id (*int*) –
- media_type (*List[str] | None*) –

messages_get_important_messages(*count=20*, *offset=None*, *start_message_id=None*, *preview_length=None*, *fields=None*, *extended=None*)

Параметры

- count (int, default: 20) – Максимальное число результатов, которые нужно получить.
- offset (Optional[int], default: None) – Смещение, необходимое для выборки определенного подмножества результатов.
- start_message_id (Optional[int], default: None) – Идентификатор сообщения, начиная с которого нужно возвращать список.
- preview_length (Optional[int], default: None) –
- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.
- extended (Optional[int], default: None) – Возвращать дополнительные поля.

messages_get_intent_users(*intent*, *subscribe_id=None*, *offset=None*, *count=20*, *extended=None*, *name_case=None*, *fields=None*)

Параметры

- intent (str) – Тип интента, который требует подписку.
- subscribe_id (Optional[int], default: None) – ID подписки, необходимый для confirmed_notification.
- offset (Optional[int], default: None) – Смещение, необходимое для выборки определенного подмножества.
- count (int, default: 20) – Количество подписчиков, информацию о которых необходимо получить.
- extended (Optional[int], default: None) – Возвращать дополнительные поля.
- name_case (Optional[List[str]], default: None) – падеж для склонения имени и фамилии пользователя.
- fields (*List[str] | None*) –

Возможные значения: именительный – nom, родительный – gen, дательный – dat, винительный – acc, творительный – ins, предложный – abl. По умолчанию nom.

Параметры

- `fields` (`Optional[List[str]]`, default: `None`) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.
- `intent` (*str*) –
- `subscribe_id` (*int / None*) –
- `offset` (*int / None*) –
- `count` (*int*) –
- `extended` (*int / None*) –
- `name_case` (*List[str] / None*) –

`messages_get_invite_link(`*peer_id*, *reset=None*`)`

Параметры

- `peer_id` (`int`) – Идентификатор назначения.
- `reset` (`Optional[int]`, default: `None`) – Сгенерировать новую ссылку, сбросив предыдущую.

`messages_get_longpoll_history(`*ts*, *pts*, *preview_length=None*, *onlines=None*, *fields=None*, *events_limit=1000*, *msgs_limit=200*, *max_msg_id=None*, *lp_version=3*, *last_n=0*, *credentials=None*`)`

Параметры

- `ts` (`int`) – Последнее значение параметра ts.
- `pts` (`int`) – последнее значение параметра new_pts.
- `preview_length` (`Optional[int]`, default: `None`) – Количество символов, по которому нужно обрезать сообщение.
- `onlines` (`Optional[int]`, default: `None`) – Возвращать в числе прочих события 8 и 9 (пользователь стал онлайн/оффлайн).
- `fields` (`Optional[List[str]]`, default: `None`) – Список дополнительных полей профилей, которые необходимо вернуть.
- `events_limit` (`int`, default: `1000`) – Лимит по количеству всех событий в истории.
- `msgs_limit` (*int*) –
- `max_msg_id` (*int / None*) –
- `lp_version` (*int*) –
- `last_n` (*int*) –
- `credentials` (*int / None*) –

Обратите внимание, параметры events_limit и msgs_limit применяются совместно. Число результатов в ответе ограничивается первым достигнутым лимитом.

Параметры

- msgs_limit (int, default: 200) – Лимит по количеству событий с сообщениями в истории.
- max_msg_id (Optional[int], default: None) – Максимальный идентификатор сообщения среди уже имеющихся в локальной копии.
- lp_version (int, default: 3) – Версия Long Poll.
- last_n (int, default: 0) –
- credentials (Optional[int], default: None) –
- ts (*int*) –
- pts (*int*) –
- preview_length (*int / None*) –
- onlines (*int / None*) –
- fields (*List[str] / None*) –
- events_limit (*int*) –

messages_get_longpoll_server(*need_pts=None*, *lp_version=3*)

Параметры

- need_pts (Optional[int], default: None) – Возвращать поле pts, необходимое для работы метода messages.getLongPollHistory.
- lp_version (int, default: 3) – Версия Long Poll.

messages_is_messages_from_group_allowed(*user_id*)

Параметры

user_id (int) – Идентификатор пользователя.

messages_mark_as_answered_conversation(*peer_id*, *answered=1*)

Параметры

- peer_id (int) – Идентификатор беседы.
- answered (int, default: 1) – Беседа отмечена отвеченной.

messages_mark_as_important_conversation(*peer_id*, *important=1*)

Параметры

- peer_id (int) – Идентификатор беседы
- important (int, default: 1) – Если сообщения необходимо пометить, как важные.

messages_mark_as_read(*message_ids=None*, *peer_id=None*, *start_message_id=None*, *mark_conversation_as_read=None*)

Параметры

- message_ids (Optional[List[int]], default: None) – Идентификаторы сообщений.
- peer_id (Optional[str], default: None) – Идентификатор назначения.
- start_message_id (Optional[int], default: None) – При передаче этого параметра будут помечены как прочитанные все сообщения, начиная с данного.

- mark_conversation_as_read (Optional[int], default: None) –

messages_pin(*peer_id*, *message_id*, *conversation_message_id*)

Параметры

- peer_id (int) – Идентификатор назначения.
- message_id (int) – Идентификатор сообщения, которое нужно закрепить.
- conversation_message_id (int) – Идентификатор сообщения беседы, которое нужно закрепить.

messages_remove_chat_user(*chat_id*, *user_id=None*, *member_id=None*)

Параметры

- chat_id (int) – Идентификатор беседы.
- user_id (Optional[int], default: None) – Идентификатор пользователя, которого необходимо исключить из беседы.
- member_id (Optional[int], default: None) – Идентификатор участника, которого необходимо исключить из беседы.

Для сообществ — идентификатор сообщества со знаком «минус».

messages_restore(*message_id*)

Параметры

message_id (int) – Идентификатор сообщения, которое нужно восстановить.

messages_search(*q=None*, *peer_id=None*, *date=None*, *preview_length=0*, *offset=None*, *count=20*, *extended=None*, *fields=None*)

Параметры

- q (Optional[str], default: None) – Подстрока, по которой будет производиться поиск.
- peer_id (Optional[int], default: None) – Фильтр по идентификатору назначения для поиска по отдельному диалогу.
- date (Optional[int], default: None) – Дата в формате DDMMYYYY. Если параметр задан, в ответе будут только сообщения, отправленные до указанной даты.
- preview_length (int, default: 0) – Количество символов, по которому нужно обрезать сообщение.
- offset (Optional[int], default: None) – Смещение, необходимое для выборки определенного подмножества сообщений из списка найденных.
- count (int, default: 20) – Количество сообщений, которое необходимо получить.
- extended (Optional[int], default: None) – Возвращать дополнительные поля для пользователей и сообществ.
- fields (Optional[List[str]], default: None) – Список ыудополнительных полей для пользователей и сообществ.

messages_search_conversations(*q*, *count=20*, *extended=None*, *fields=None*)

Параметры

- q (str) – Поисковой запрос.
- count (int, default: 20) – Максимальное число результатов для получения.
- extended (Optional[int], default: None) – Возвращать дополнительные поля.
- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.

messages_send(*user_id=None*, *peer_id=None*, *peer_ids=None*, *domain=None*, *chat_id=None*, *message=None*, *lat=None*, *long=None*, *attachment=None*, *reply_to=None*, *forward_messages=None*, *forward=None*, *sticker_id=None*, *keyboard=None*, *template=None*, *payload=None*, *content_source=None*, *dont_parse_links=None*, *disable_mentions=None*, *intent='default'*, *subscribe_id=None*)

Параметры

- user_id (Optional[int], default: None) – Идентификатор пользователя, которому отправляется сообщение.
- peer_id (Optional[int], default: None) – Идентификатор назначения.
- peer_ids (Optional[List[int]], default: None) – Идентификаторы получателей сообщения (при необходимости отправить сообщение сразу нескольким пользователям)
- domain (Optional[str], default: None) – Короткий адрес пользователя.
- chat_id (Optional[int], default: None) – Идентификатор беседы, к которой будет относиться сообщение.
- message (Optional[str], default: None) – Текст личного сообщения. Обязательный параметр, если не задан параметр attachment.
- lat (Optional[float], default: None) – Географическая широта (от -90 до 90).
- long (Optional[float], default: None) – Географическая долгота (от -180 до 180).
- attachment (default: None) – Медиавложения к личному сообщению, перечисленные через запятую.
- reply_to (Optional[int], default: None) – Идентификатор сообщения, на которое требуется ответить.
- forward_messages (Optional[List[int]], default: None) – Идентификаторы пересылаемых сообщений.
- forward (default: None) – JSON-объект.
- sticker_id (Optional[int], default: None) – Идентификатор стикера.
- keyboard (default: None) – Объект, описывающий клавиатуру бота.
- template (Optional[Dict], default: None) – Объект, описывающий шаблон сообщения.
- payload (default: None) – Полезная нагрузка.
- content_source (Optional[Dict], default: None) – Объект, описывающий источник пользовательского контента для чат-ботов.

- `dont_parse_links` (`Optional[int]`, default: `None`) – Не создавать сниппет ссылки из сообщения.
- `disable_mentions` (`Optional[int]`, default: `None`) – Отключить уведомление об упоминании в сообщении.
- `intent` (`str`, default: `'default'`) – Строка, описывающая интенты.
- `subscribe_id` (`Optional[int]`, default: `None`) –

`messages_send_message_event_answer(`*event_id*, *user_id*, *peer_id*, *event_data=None*`)`

Параметры

- `event_id` (`str`) – Случайная строка, которая возвращается в событии message_event.
- `user_id` (`int`) – Идентификатор пользователя.
- `peer_id` (`int`) – Идентификатор диалога со стороны сообщества.
- `event_data` (default: `None`) – Объект действия, которое должно произойти после нажатия на кнопку.

`messages_set_activity(`*user_id*, *type*, *peer_id*`)`

Параметры

- `user_id` (`int`) – Идентификатор пользователя.
- `type` (`str`) – typing — пользователь начал набирать текст, audiomessage — пользователь записывает голосовое сообщение.
- `peer_id` (`int`) – Идентификатор назначения.

`messages_set_chat_photo(`*file*`)`

Параметры

`file` (`str`) – Содержимое поля response из ответа специального upload сервера, полученного в результате загрузки изображения на адрес, полученный методом photos.getChatUploadServer.

`messages_unpin(`*peer_id*`)`

Параметры

`peer_id` (`int`) – Идентификатор назначения.

`groups_add_address(`*title*, *address*, *country_id*, *city_id*, *latitude*, *longitude*, *metro_id=None*, *additional_address=None*, *phone=None*, *work_info_status=None*, *timetable=None*, *is_main_address=None*`)`

Параметры

- `title` (`str`) – Заголовок адреса.
- `address` (`str`) – Строка адреса.
- `additional_address` (`Optional[str]`, default: `None`) – Дополнительное описание адреса.
- `country_id` (`int`) – Идентификатор страны.
- `city_id` (`int`) – Идентификатор города.
- `latitude` (`float`) – Географическая широта отметки, заданная в градусах (от -90 до 90).

- `longitude` (`float`) – Географическая долгота отметки, заданная в градусах (от -180 до 180)
- `metro_id` (`Optional[int]`, default: `None`) – Идентификатор станции метро.
- `phone` (`Optional[str]`, default: `None`) – Номер телефона.
- `work_info_status` (`Optional[str]`, default: `None`) – тип расписания. Возможные значения: no_information – нет информации о расписании; temporarily_closed — временно закрыто; always_opened — открыто круглосуточно; forever_closed — закрыто навсегда; timetable — открыто в указанные часы работы. Для этого типа расписания необходимо передать параметр timetable;
- `timetable` (default: `None`) –
- `is_main_address` (`Optional[int]`, default: `None`) – Установить адрес основным.

`groups_delete_address(`*address_id*`)`

Параметры

`address_id` (`int`) – Идентификатор адреса.

`groups_disable_online()`

`groups_edit(`*title=None*, *description=None*, *screen_name=None*, *access=None*, *website=None*, *subject=None*, *email=None*, *phone=None*, *rss=None*, *event_start_date=None*, *event_finish_date=None*, *event_group_id=None*, *public_category=None*, *public_subcategory=None*, *public_date=None*, *wall=None*, *topics=None*, *photos=None*, *video=None*, *audio=None*, *links=None*, *events=None*, *places=None*, *contacts=None*, *docs=None*, *wiki=None*, *messages=None*, *articles=None*, *addresses=None*, *age_limits=None*, *market=None*, *market_comments=None*, *market_country=None*, *market_city=None*, *market_currency=None*, *market_contact=None*, *market_wiki=None*, *obscene_filter=None*, *obscene_stopwords=None*, *obscene_words=None*, *main_section=None*, *secondary_section=None*, *country=None*, *city=None*`)`

Параметры

- `title` (`Optional[str]`, default: `None`) – Название сообщества.
- `description` (`Optional[str]`, default: `None`) – Описание сообщества.
- `screen_name` (`Optional[str]`, default: `None`) – Короткое имя сообщества.
- `access` (`Optional[int]`, default: `None`) – тип группы. Возможные значения: 0 – открытая; 1 – закрытая; 2 – частная;
- `website` (`Optional[str]`, default: `None`) – Адрес сайта, который будет указан в информации о группе.
- `subject` (`Optional[int]`, default: `None`) – Тематика сообщества.
- `email` (`Optional[str]`, default: `None`) – Электронный адрес организатора (для мероприятий).
- `phone` (`Optional[str]`, default: `None`) – Номер телефона организатора (для мероприятий).
- `rss` (`Optional[str]`, default: `None`) – Адрес rss для импорта новостей
- `event_start_date` (`Optional[int]`, default: `None`) – Дата начала события.
- `event_finish_date` (`Optional[int]`, default: `None`) – Дата окончания события.

- `event_group_id` (`Optional[int]`, default: `None`) – идентификатор группы, которая является организатором события (только для событий).
- `public_category` (`Optional[int]`, default: `None`) – Категория публичной страницы.
- `public_subcategory` (`Optional[int]`, default: `None`) – Подкатегория публичной станицы.
- `public_date` (`Optional[str]`, default: `None`) – дата основания компании, организации, которой посвящена публичная страница в виде строки формата «dd.mm.YYYY».
- `wall` (`Optional[int]`, default: `None`) – Стена. Возможные значения: 0 — выключена; 1 — открытая; 2 — ограниченная (доступно только для групп и событий); 3 — закрытая (доступно только для групп и событий).
- `topics` (`Optional[int]`, default: `None`) – Обсуждения. Возможные значения: 0 — выключены; 1 — открытые; 2 — ограниченные (доступно только для групп и событий).
- `photos` (`Optional[int]`, default: `None`) – Фотографии. Возможные значения: 0 — выключены; 1 — открытые; 2 — ограниченные (доступно только для групп и событий).
- `video` (`Optional[int]`, default: `None`) – Видеозаписи. Возможные значения: 0 — выключены; 1 — открытые; 2 — ограниченные (доступно только для групп и событий).
- `audio` (`Optional[int]`, default: `None`) – Аудиозаписи. Возможные значения: 0 — выключены; 1 — открытые; 2 — ограниченные (доступно только для групп и событий).
- `links` (`Optional[int]`, default: `None`) – Ссылки (доступно только для публичных страниц). Возможные значения: 0 — выключены; 1 — включены.
- `events` (`Optional[int]`, default: `None`) – События (доступно только для публичных страниц). Возможные значения: 0 — выключены; 1 — включены.
- `places` (`Optional[int]`, default: `None`) – Места (доступно только для публичных страниц). Возможные значения: 0 — выключены; 1 — включены.
- `contacts` (`Optional[int]`, default: `None`) – Контакты (доступно только для публичных страниц). Возможные значения: 0 — выключены; 1 — включены.
- `docs` (`Optional[int]`, default: `None`) – Документы сообщества. Возможные значения: 0 — выключены; 1 — открытые; 2 — ограниченные (доступно только для групп и событий).
- `wiki` (`Optional[int]`, default: `None`) – wiki-материалы сообщества. Возможные значения: 0 — выключены; 1 — открытые; 2 — ограниченные (доступно только для групп и событий).
- `messages` (`Optional[int]`, default: `None`) – Сообщения сообщества. Возможные значения: 0 — выключены; 1 — включены.
- `articles` (`Optional[int]`, default: `None`) –
- `addresses` (`Optional[int]`, default: `None`) –
- `age_limits` (`Optional[int]`, default: `None`) – Возрастное ограничение для сообщества. Возможные значения: 1 — нет ограничений; 2 — 16+; 3 — 18+.

- market (Optional[int], default: None) – Товары. Возможные значения: 0 — выключены; 1 — включены.
- market_comments (Optional[int], default: None) – Комментарии к товарам. Возможные значения: 0 — выключены; 1 — включены.
- market_country (Optional[List[int]], default: None) – Регионы доставки товаров.
- market_city (Optional[List[int]], default: None) – Города доставки товаров (в случае если указана одна страна).
- market_currency (Optional[int], default: None) – Идентификатор валюты магазина. Возможные значения: 643 — российский рубль; 980 — украинская гривна; 398 — казахстанский тенге; 978 — евро; 840 — доллар США.
- market_contact (Optional[int], default: None) – Контакт для связи для продавцом.
- market_wiki (Optional[int], default: None) – Идентификатор wiki-страницы с описанием магазина.
- obscene_filter (Optional[int], default: None) – Фильтр нецензурных выражений в комментариях. Возможные значения: 0 — выключен; 1 — включен.
- obscene_stopwords (Optional[int], default: None) – Фильтр по ключевым словам в комментариях. Возможные значения: 0 — выключен; 1 — включен.
- obscene_words (Optional[List[str]], default: None) – Ключевые слова для фильтра комментариев.
- main_section (Optional[int], default: None) –
- secondary_section (Optional[int], default: None) –
- country (Optional[int], default: None) –
- city (Optional[int], default: None) –

groups_edit_address(*address_id*, *title=None*, *address=None*, *additional_address=None*, *country_id=None*, *city_id=None*, *metro_id=None*, *latitude=None*, *longitude=None*, *phone=None*, *work_info_status=None*, *timetable=None*, *is_main_address=None*)

Параметры

- address_id (int) – Идентификатор адреса.
- title (Optional[str], default: None) – Заголовок адреса.
- address (Optional[str], default: None) – Строка адреса. *Невский проспект, дом 28*
- additional_address (Optional[str], default: None) – Дополнительное описание адреса.
- country_id (Optional[int], default: None) – Идентификатор страны.
- city_id (Optional[int], default: None) – Идентификатор города.
- metro_id (Optional[int], default: None) – Идентификатор станции метро.
- latitude (Optional[float], default: None) – Географическая широта отметки, заданная в градусах (от -90 до 90).

- `longitude` (`Optional[float]`, default: `None`) – Географическая долгота отметки, заданная в градусах (от -180 до 180).
- `phone` (`Optional[str]`, default: `None`) – Номер телефона.
- `work_info_status` (`Optional[str]`, default: `None`) – Тип расписания.
- `timetable` (default: `None`) –
- `is_main_address` (`Optional[int]`, default: `None`) – Установить адрес основным.

`groups_enable_online()`

`groups_get_by_id(`*fields=None*`)`

Параметры

`fields` (`Optional[List[str]]`, default: `None`) – Список дополнительных полей, которые необходимо вернуть.

`groups_get_members(`*sort='id_asc'*, *offset=0*, *count=1000*, *fields=None*`)`

Параметры

- `sort` (`str`, default: `'id_asc'`) – Сортировка, с которой необходимо вернуть список участников. Возможные значения: id_asc — в порядке возрастания id; id_desc — в порядке убывания id; time_asc — в хронологическом порядке по вступлению в сообщество; time_desc — в антихронологическом порядке по вступлению в сообщество.
- `offset` (`int`, default: `0`) – Смещение, необходимое для выборки определенного подмножества участников.
- `count` (`int`, default: `1000`) – Количество участников сообщества, информацию о которых необходимо получить.
- `fields` (`Optional[List[str]]`, default: `None`) – Список дополнительных полей, которые необходимо вернуть.

`groups_is_member(`*user_id=None*, *user_ids=None*`)`

Параметры

- `user_id` (`Optional[int]`, default: `None`) – Идентификатор пользователя.
- `user_ids` (`Optional[List[int]]`, default: `None`) – Идентификаторы пользователей, не более 500.

`groups_get_banned(`*fields=None*, *count=20*, *owner_id=None*, *offset=None*`)`

Параметры

- `fields` (`Optional[List[str]]`, default: `None`) – Смещение, необходимое для выборки определенного подмножества черного списка.
- `count` (`int`, default: `20`) – Количество пользователей, которое необходимо вернуть.
- `owner_id` (`Optional[int]`, default: `None`) – Идентификатор пользователя или сообщества из чёрного списка, информацию о котором нужно получить.
- `offset` (`Optional[int]`, default: `None`) – Смещение, необходимое для выборки определенного подмножества черного списка.

`groups_get_online_status()`

groups_get_token_permissions()

groups_set_settings(*messages=None*, *bots_capabilities=None*, *bots_start_button=None*, *bots_add_to_chat=None*)

Параметры

- messages (Optional[int], default: None) – Сообщения сообщества. Возможные значения: 0 — выключены; 1 — включены.
- bots_capabilities (Optional[int], default: None) – Возможности ботов (использование клавиатуры, добавление в беседу). Возможные значения: 0 — выключены; 1 — включены.
- bots_start_button (Optional[int], default: None) – Кнопка «Начать» в диалоге с сообществом. Работает, в случае если bots_capabilities=1. Возможные значения: 0 — выключена; 1 — включена.
- bots_add_to_chat (Optional[int], default: None) – Добавление бота в беседы. Работает, в случае если bots_capabilities=1. Возможные значения: 0 — запрещено; 1 — разрешено.

groups_get_longpoll_server()

board_delete_comment(*topic_id*, *comment_id*)

Параметры

- topic_id (int) – Идентификатор обсуждения.
- comment_id (int) – Идентификатор комментария в обсуждении.

board_restore_comment(*topic_id*, *comment_id*)

Параметры

- topic_id (int) – Идентификатор обсуждения.
- comment_id (int) – Идентификатор комментария.

docs_get_messages_upload_server(*peer_id*, *type='doc'*)

Параметры

- peer_id (int) – Идентификатор назначения.
- type (str, default: 'doc') – Тип документа. Возможные значения: doc — обычный документ; audio_message — голосовое сообщение;

docs_get_wall_upload_server()

docs_save(*file*, *title=None*, *tags=None*, *return_tags=None*)

Параметры

- file – Параметр, возвращаемый в результате загрузки файла на сервер.
- title (Optional[str], default: None) – Название документа.
- tags (Optional[List[str]], default: None) – Метки для поиска.
- return_tags (Optional[int], default: None) –

docs_search(*q*, *search_own=None*, *count=20*, *offset=None*, *return_tags=None*)

Параметры

- q (str) – Строка поискового запроса. Например, зеленые тапочки.
- search_own (Optional[int], default: None) – 1 — искать среди собственных документов пользователя.
- count (int, default: 20) – Количество документов, информацию о которых нужно вернуть.
- offset (Optional[int], default: None) – Смещение, необходимое для выборки определенного подмножества документов.
- return_tags (Optional[int], default: None) –

market_edit_order(*user_id*, *order_id*, *merchant_comment=None*, *status=None*, *track_number=None*, *payment_status=None*, *delivery_price=None*, *width=None*, *length=None*, *height=None*, *weight=None*, *comment_for_user=None*)

Параметры

- user_id (int) – Идентификатор пользователя.
- order_id (int) – Идентификатор заказа.
- merchant_comment (Optional[str], default: None) – Комментарий продавца.
- status (Optional[int], default: None) – Статус заказа. Возможные значения:

  0 - новый; 1 - согласуется; 2 - собирается; 3 - доставляется; 4 - выполнен; 5 - отменен; 6 - возвращен.
- track_number (Optional[str], default: None) – Трек-номер.
- payment_status (Optional[str], default: None) – Статус платежа. Возможные значения:

  not_paid - не оплачен; paid - оплачен; returned - возвращен.
- delivery_price (Optional[int], default: None) – Стоимость доставки.
- width (Optional[float], default: None) – Ширина.
- length (Optional[float], default: None) – Длина.
- height (Optional[float], default: None) – Высота.
- weight (Optional[float], default: None) – Вес.
- comment_for_user (Optional[str], default: None) –

market_get_group_orders(*offset=None*, *count=10*)

Параметры

- offset (Optional[int], default: None) – Смещение относительно первого найденного заказа для выборки определенного подмножества.
- count (int, default: 10) – Количество возвращаемых заказов.

market_get_order_by_id(*order_id*, *user_id=None*, *extended=None*)

Параметры

- order_id (int) – Идентификатор заказа.
- user_id (Optional[int], default: None) – Идентификатор пользователя.
- extended (Optional[int], default: None) –

market_get_order_items(*order_id*, *user_id=None*, *offset=None*, *count=50*)

Параметры

- order_id (int) – Идентификатор заказа.
- user_id (Optional[int], default: None) – Id пользователя, который сделал заказ.
- offset (Optional[int], default: None) – Смещение относительно первого найденного товара в заказе для выборки определенного подмножества.
- count (int, default: 50) – Количество возвращаемых товаров в заказе.

photos_get_chat_upload_server(*chat_id*, *crop_x=None*, *crop_y=None*, *crop_width=None*)

Параметры

- chat_id (int) – Идентификатор беседы, для которой нужно загрузить фотографию.
- crop_x (Optional[float], default: None) – Координата x для обрезки фотографии (верхний правый угол).
- crop_y (Optional[float], default: None) – Координата y для обрезки фотографии (верхний правый угол).
- crop_width (Optional[int], default: None) – Ширина фотографии после обрезки в px.

photos_get_messages_upload_server()

photos_get_owner_cover_photo_upload_server(*crop_x=None*, *crop_y=None*, *crop_x2=None*, *crop_y2=None*)

Параметры

- crop_x (Optional[float], default: None) – Координата X верхнего левого угла для обрезки изображения.
- crop_y (Optional[float], default: None) – Координата Y верхнего левого угла для обрезки изображения.
- crop_x2 (Optional[float], default: None) – Координата X нижнего правого угла для обрезки изображения.
- crop_y2 (Optional[float], default: None) – Координата Y нижнего правого угла для обрезки изображения.

photos_save_messages_photo(*photo*, *server=None*, *hash=None*)

Параметры

- photo (str) – Параметр, возвращаемый в результате загрузки фотографии на сервер.
- server (Optional[int], default: None) – Параметр, возвращаемый в результате загрузки фотографии на сервер.
- hash (Optional[str], default: None) – Параметр, возвращаемый в результате загрузки фотографии на сервер.

photos_save_owner_cover_photo(*hash*, *photo*)

Параметры

- **hash** (str) – Параметр hash, полученный в результате загрузки фотографии на сервер.
- **photo** (str) – Параметр photo, полученный в результате загрузки фотографии на сервер.

podcasts_search_podcast(*search_string*, *offset=None*, *count=20*)

Параметры

- **search_string** (str) –
- **offset** (Optional[int], default: None) –
- **count** (int, default: 20) –

storage_get(*key=None*, *keys=None*, *user_id=None*)

Параметры

- **key** (Optional[str], default: None) – Название переменной.
- **keys** (Optional[List[str]], default: None) – Список названий переменных. Если указано, параметр key не учитывается.
- **user_id** (Optional[int], default: None) – id пользователя, переменная которого устанавливается, в случае если данные запрашиваются серверным методом.

storage_get_keys(*user_id*, *offset=None*, *count=100*)

Параметры

- **user_id** (int) – id пользователя, названия переменных которого получаются, в случае если данные запрашиваются серверным методом.
- **offset** (Optional[int], default: None) – Смещение, необходимое для выборки определенного подмножества названий переменных
- **count** (int, default: 100) – Количество названий переменных, информацию о которых необходимо получить.

Тип результата
List

storage_set(*key*, *value=None*, *user_id=None*)

Параметры

- **key** (str) – Название переменной. Может содержать символы латинского алфавита, цифры, знак тире, нижнее подчёркивание [a-zA-Z_-0-9].
- **value** (Optional[str], default: None) – Значение переменной, сохраняются только первые 4096 байта.
- **user_id** (Optional[int], default: None) – id пользователя, переменная которого устанавливается, в случае если данные запрашиваются серверным методом.

users_get(*user_ids*, *fields=None*, *name_case='nom'*)

Параметры

- **user_ids** (List[int]) – Идентификаторы пользователей

- fields (Optional[List[str]], default: None) – Список дополнительных полей профилей, которые необходимо вернуть.
- name_case (str, default: 'nom') – Падеж для склонения имени и фамилии пользователя. Возможные значения:

  Именительный – nom, Родительный – gen, Дательный – dat, Винительный – acc, Творительный – ins, Предложный – abl.

stories_delete(*owner_id*, *story_id*, *stories=None*)

Параметры

- owner_id (int) – Идентификатор владельца истории.
- story_id (int) – Идентификатор истории.
- stories (Optional[List[str]], default: None) –

stories_get(*owner_id*, *extended=None*, *fields=None*)

Параметры

- owner_id (int) – Идентификатор пользователя, истории которого необходимо получить.
- extended (Optional[int], default: None) – 1 — возвращать в ответе дополнительную информацию о профилях пользователей.
- fields (Optional[List[str]], default: None) –

stories_get_by_id(*stories*, *extended=None*, *fields=None*)

Параметры

- stories (List[int]) – Идентификаторы историй.
- extended (Optional[int], default: None) – 1 — возвращать в ответе дополнительную информацию о пользователях.
- fields (Optional[List[str]], default: None) – Дополнительные поля профилей и сообществ, которые необходимо вернуть в ответе.

stories_get_photo_upload_server(*add_to_news=None*, *user_ids=None*, *reply_to_story=None*, *link_text=None*, *link_url=None*, *clickable_stickers=None*)

Параметры

- add_to_news (Optional[int], default: None) – 1 — разместить историю в новостях. Обязательно, если не указан user_ids.
- user_ids (Optional[List[int]], default: None) – Идентификаторы пользователей, которые будут видеть историю (для отправки в личном сообщении). Обязательно, если add_to_news не передан.
- reply_to_story (Optional[str], default: None) – Идентификатор истории, в ответ на которую создается новая.
- link_text (Optional[str], default: None) – Текст ссылки для перехода из истории.
- link_url (Optional[str], default: None) – Адрес ссылки для перехода из истории. Допустимы только внутренние ссылки https://vk.com.
- clickable_stickers (default: None) – Объект кликабельного стикера.

stories_get_replies(*owner_id*, *story_id*, *access_key=None*, *extended=None*, *fields=None*)

Параметры

- owner_id (int) – Идентификатор владельца истории.
- story_id (int) – Идентификатор истории.
- access_key (Optional[str], default: None) – Ключ доступа для приватного объекта.
- extended (Optional[int], default: None) – 1 — возвращать дополнительную информацию о профилях и сообществах.
- fields (Optional[List[str]], default: None) – Дополнительные поля профилей и сообществ, которые необходимо вернуть в ответе.

stories_get_stats(*owner_id*, *story_id*)

Параметры

- owner_id (int) – Идентификатор владельца истории.
- story_id (int) – Идентификатор истории.

stories_get_video_upload_server(*user_ids=None*, *add_to_news=None*, *reply_to_story=None*, *link_text=None*, *link_url=None*, *clickable_stickers=None*)

Параметры

- user_ids (Optional[List[int]], default: None) – Идентификаторы пользователей, которые будут видеть историю (для отправки в личном сообщении).
- add_to_news (Optional[int], default: None) – 1 — разместить историю в новостях.
- reply_to_story (Optional[str], default: None) – Идентификатор истории, в ответ на которую создается новая.
- link_text (Optional[str], default: None) – Текст ссылки для перехода из истории.
- link_url (Optional[str], default: None) – Адрес ссылки для перехода из истории.
- clickable_stickers (default: None) – Объект кликабельного стикера.

stories_get_viewers(*owner_id*, *story_id*, *count=100*, *offset=None*, *extended=None*)

Параметры

- owner_id (int) – Идентификатор владельца истории.
- story_id (int) – Идентификатор истории.
- count (int, default: 100) – Максимальное число результатов в ответе.
- offset (Optional[int], default: None) – Сдвиг для получения определённого подмножества результатов.
- extended (Optional[int], default: None) – 1 — возвращать в ответе расширенную информацию о пользователях.

stories_hide_all_replies(*owner_id*)

Параметры
owner_id (int) – Идентификатор пользователя, ответы от которого нужно скрыть.

stories_hide_reply(*owner_id*, *story_id*)

Параметры

- owner_id (int) – Идентификатор владельца истории (ответной).
- story_id (int) – Идентификатор истории (ответной).

stories_save(*upload_results*, *extended=None*, *fields=None*)

Параметры

- upload_results (List[str]) – Список строк, которые возвращает stories.getPhotoUploadServer или stories.getVideoUploadServer.
- extended (Optional[int], default: None) –
- fields (Optional[List[str]], default: None) –

utils_check_link(*url*)

Параметры
url (str) – Внешняя ссылка, которую необходимо проверить.

utils_get_link_stats(*key*, *source='vk_cc'*, *access_key=None*, *interval='day'*, *intervals_count=1*, *extended=None*)

Параметры

- key (str) – Сокращенная ссылка (часть URL после «vk.cc/»).
- source (str, default: 'vk_cc') –
- access_key (Optional[str], default: None) – Ключ доступа к приватной статистике ссылки.
- interval (str, default: 'day') – Единица времени для подсчета статистики. Возможные значения: hour, day, week, month, forever.
- intervals_count (int, default: 1) – Длительность периода для получения статистики в выбранных единицах (из параметра interval).
- extended (Optional[int], default: None) – 1 — возвращать расширенную статистику (пол/возраст/страна/город), 0 — возвращать только количество переходов.

utils_get_server_time()

Возвращает число, соответствующее времени в UnixTime.

utils_get_short_link(*url*, *private=None*)

Параметры

- url (str) – URL, для которого необходимо получить сокращенный вариант.
- private (Optional[int], default: None) – 1 — статистика ссылки приватная, 0 — статистика ссылки общедоступная.

utils_resolve_screen_name(*screen_name*)

Параметры
screen_name (str) – Короткое имя пользователя, группы или приложения.

wall_close_comments(*owner_id*, *post_id*)

Параметры

- owner_id (int) –
- post_id (int) –

wall_create_comment(*post_id*, *message=None*, *reply_to_comment=None*, *attachments=None*, *sticker_id=None*)

Параметры

- post_id (int) – Идентификатор записи на стене.
- message (Optional[str], default: None) – Текст комментария. Обязательный параметр, если не передан параметр attachments.
- reply_to_comment (Optional[int], default: None) – Идентификатор комментария, в ответ на который должен быть добавлен новый комментарий.
- attachments (default: None) –
- sticker_id (Optional[int], default: None) – Идентификатор стикера.

class ApiMethod(*access_token*, *api_version*)

Параметры

- access_token (str) –
- api_version (str) –

base_url: str = 'https://api.vk.com/method/'

http = <requests.sessions.Session object>

rps_delay = 0.05

last_request = 0.0

class Vk(*access_token*, *api_version='5.126'*)

Читайте подробнее про методы https://vk.com/dev/methods

Параметры

- access_token (str) – Токен сообщества
- api_version (str, default: '5.126') – Версия Api

messages_create_chat(*user_ids*, *title*)

Параметры

- user_ids (List[int]) – Идентификаторы пользователей, которых нужно включить в мультидиалог.
- title (str) – Название беседы.

messages_delete(*message_ids=None*, *peer_id=None*, *spam=None*, *delete_for_all=None*, *cmids=None*)

Параметры

- message_ids (Optional[List[int]], default: None) – Список идентификаторов сообщений.
- spam (Optional[int], default: None) – Помечает сообщения как спам.
- delete_for_all (Optional[int], default: None) – Удаление для всех.
- cmids (Optional[List[int]], default: None) – Conversation Message Ids
- peer_id (Optional[int], default: None) –

messages_delete_chat_photo(*chat_id*)

Параметры

chat_id (int) – Идентификатор беседы.

messages_delete_conversation(*user_id=None*, *peer_id=None*)

Параметры

- user_id (Optional[int], default: None) – Идентификатор пользователя. Если требуется очистить историю беседы, используйте peer_id.
- peer_id (Optional[int], default: None) – Идентификатор назначения.

messages_edit(*peer_id*, *message=None*, *lat=None*, *long=None*, *attachment=None*, *keep_forward_messages=None*, *keep_snippets=None*, *dont_parse_links=None*, *message_id=None*, *conversation_message_id=None*, *template=None*, *keyboard=None*)

Параметры

- peer_id (int) – Идентификатор назначения.
- message (Optional[str], default: None) – Текст сообщения. Обязательный параметр, если не задан параметр attachment.
- lat (Optional[float], default: None) – Географическая широта (от -90 до 90).
- long (Optional[float], default: None) – Географическая долгота (от -180 до 180).
- attachment (default: None) – Медиавложения к личному сообщению, перечисленные через запятую.
- keep_forward_messages (Optional[int], default: None) – Сохранить прикреплённые пересланные сообщения.
- keep_snippets (Optional[int], default: None) – Сохранить прикреплённые внешние ссылки (сниппеты).
- dont_parse_links (Optional[int], default: None) – Не создавать сниппет ссылки из сообщения.
- message_id (Optional[int], default: None) – Идентификатор сообщения.
- conversation_message_id (Optional[int], default: None) – Идентификатор сообщения в беседе.
- template (default: None) – Объект, описывающий шаблоны сообщений.

- keyboard (default: None) – Объект, описывающий клавиатуру бота.

messages_edit_chat(*chat_id*, *title*)

Параметры

- chat_id (int) – Идентификатор беседы.
- title (str) – Новое название для беседы.

messages_get_by_conversation_message_id(*peer_id*, *conversation_message_ids*, *extended=None*, *fields=None*)

Параметры

- peer_id (int) – Идентификатор назначения.
- conversation_message_ids (List[int]) – Идентификаторы сообщений. Максимум 100 идентификаторов.
- extended (Optional[int], default: None) – Возвращать дополнительные поля.
- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.

messages_get_by_id(*message_ids*, *preview_length=None*, *extended=None*, *fields=None*)

Параметры

- message_ids (List[int]) – Идентификаторы сообщений. Максимум 100 идентификаторов.
- preview_length (Optional[int], default: None) – Количество символов, по которому нужно обрезать сообщение.
- extended (Optional[int], default: None) – Возвращать дополнительные поля.
- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.

messages_get_conversation_members(*peer_id*, *fields*)

Параметры

- peer_id (int) – Идентификатор назначения.
- fields (List[str]) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.

messages_get_conversations(*offset=0*, *count=20*, *filter='all'*, *extended=None*, *start_message_id=None*, *fields=None*)

Параметры

- offset (int, default: 0) – Смещение, необходимое для выборки определенного подмножества результатов.
- count (int, default: 20) – Максимальное число результатов, которые нужно получить.
- filter (str, default: 'all') – Фильтр. Возможные значения: all, unread, important, unanswered.
- extended (Optional[int], default: None) – Возвращать дополнительные поля.

- start_message_id (Optional[int], default: None) – Идентификатор сообщения, начиная с которого нужно возвращать беседы.
- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.

messages_get_conversations_by_id(*peer_ids*, *extended=None*, *fields=None*)

Параметры

- peer_ids (List[int]) – Идентификаторы назначений.
- extended (Optional[int], default: None) – Возвращать дополнительные поля.
- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.

messages_get_history(*offset=None*, *count=20*, *user_id=None*, *peer_id=None*, *start_message_id=None*, *rev=None*, *extended=None*, *fields=None*)

Параметры

- offset (Optional[int], default: None) – Смещение, необходимое для выборки определенного подмножества сообщений
- count (int, default: 20) – Количество сообщений, которое необходимо получить (но не более 200)
- user_id (Optional[int], default: None) – Идентификатор пользователя, историю переписки с которым необходимо вернуть.
- peer_id (Optional[int], default: None) – Идентификатор назначения.
- start_message_id (Optional[int], default: None) – Идентификатор сообщения, начиная с которого нужно возвращать беседы.
- rev (Optional[int], default: None) – Возвращать сообщения в хронологическом порядке (по умолчанию).
- extended (Optional[int], default: None) – Возвращать дополнительные поля.
- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.

messages_get_history_attachments(*peer_id*, *media_type=None*, *start_from=None*, *count=20*, *photo_sizes=None*, *fields=None*, *preserve_order=None*, *max_forwards_level=45*)

Параметры

- peer_id (int) – Идентификатор назначения.
- media_type (Optional[List[str]], default: None) – Тип материалов, который необходимо вернуть.
- start_from (*str | None*) –
- count (*int*) –
- photo_sizes (*int | None*) –
- fields (*List[str] | None*) –
- preserve_order (*int | None*) –
- max_forwards_level (*int*) –

Доступные значения: photo, video, audio, doc, link, market, wall, share.

Параметры

- `start_from` (`Optional[str]`, default: `None`) – Смещение, необходимое для выборки определенного подмножества объектов.
- `count` (`int`, default: `20`) – Количество объектов, которое необходимо получить (но не более 200).
- `photo_sizes` (`Optional[int]`, default: `None`) – Параметр, указывающий нужно ли возвращать ли доступные размеры фотографии в специальном формате.
- `fields` (`Optional[List[str]]`, default: `None`) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.
- `preserve_order` (`Optional[int]`, default: `None`) – Параметр, указывающий нужно ли возвращать вложения в оригинальном порядке.
- `max_forwards_level` (`int`, default: `45`) – Максимальная глубина вложенности пересланных сообщений.
- `peer_id` (*int*) –
- `media_type` (*List[str] | None*) –

`messages_get_important_messages`(*count=20*, *offset=None*, *start_message_id=None*, *preview_length=None*, *fields=None*, *extended=None*)

Параметры

- `count` (`int`, default: `20`) – Максимальное число результатов, которые нужно получить.
- `offset` (`Optional[int]`, default: `None`) – Смещение, необходимое для выборки определенного подмножества результатов.
- `start_message_id` (`Optional[int]`, default: `None`) – Идентификатор сообщения, начиная с которого нужно возвращать список.
- `preview_length` (`Optional[int]`, default: `None`) –
- `fields` (`Optional[List[str]]`, default: `None`) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.
- `extended` (`Optional[int]`, default: `None`) – Возвращать дополнительные поля.

`messages_get_intent_users`(*intent*, *subscribe_id=None*, *offset=None*, *count=20*, *extended=None*, *name_case=None*, *fields=None*)

Параметры

- `intent` (`str`) – Тип интента, который требует подписку.
- `subscribe_id` (`Optional[int]`, default: `None`) – ID подписки, необходимый для confirmed_notification.
- `offset` (`Optional[int]`, default: `None`) – Смещение, необходимое для выборки определенного подмножества.
- `count` (`int`, default: `20`) – Количество подписчиков, информацию о которых необходимо получить.
- `extended` (`Optional[int]`, default: `None`) – Возвращать дополнительные поля.

- name_case (Optional[List[str]], default: None) – падеж для склонения имени и фамилии пользователя.
- fields (*List[str] / None*) –

Возможные значения: именительный – nom, родительный – gen, дательный – dat, винительный – acc, творительный – ins, предложный – abl. По умолчанию nom.

Параметры

- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.
- intent (*str*) –
- subscribe_id (*int / None*) –
- offset (*int / None*) –
- count (*int*) –
- extended (*int / None*) –
- name_case (*List[str] / None*) –

messages_get_invite_link(*peer_id*, *reset=None*)

Параметры

- peer_id (int) – Идентификатор назначения.
- reset (Optional[int], default: None) – Сгенерировать новую ссылку, сбросив предыдущую.

messages_get_longpoll_history(*ts*, *pts*, *preview_length=None*, *onlines=None*, *fields=None*, *events_limit=1000*, *msgs_limit=200*, *max_msg_id=None*, *lp_version=3*, *last_n=0*, *credentials=None*)

Параметры

- ts (int) – Последнее значение параметра ts.
- pts (int) – последнее значение параметра new_pts.
- preview_length (Optional[int], default: None) – Количество символов, по которому нужно обрезать сообщение.
- onlines (Optional[int], default: None) – Возвращать в числе прочих события 8 и 9 (пользователь стал онлайн/оффлайн).
- fields (Optional[List[str]], default: None) – Список дополнительных полей профилей, которые необходимо вернуть.
- events_limit (int, default: 1000) – Лимит по количеству всех событий в истории.
- msgs_limit (*int*) –
- max_msg_id (*int / None*) –
- lp_version (*int*) –
- last_n (*int*) –
- credentials (*int / None*) –

Обратите внимание, параметры events_limit и msgs_limit применяются совместно. Число результатов в ответе ограничивается первым достигнутым лимитом.

Параметры

- `msgs_limit` (`int`, default: `200`) – Лимит по количеству событий с сообщениями в истории.
- `max_msg_id` (`Optional[int]`, default: `None`) – Максимальный идентификатор сообщения среди уже имеющихся в локальной копии.
- `lp_version` (`int`, default: `3`) – Версия Long Poll.
- `last_n` (`int`, default: `0`) –
- `credentials` (`Optional[int]`, default: `None`) –
- `ts` (*int*) –
- `pts` (*int*) –
- `preview_length` (*int / None*) –
- `onlines` (*int / None*) –
- `fields` (*List[str] / None*) –
- `events_limit` (*int*) –

`messages_get_longpoll_server`(*need_pts=None*, *lp_version=3*)

Параметры

- `need_pts` (`Optional[int]`, default: `None`) – Возвращать поле pts, необходимое для работы метода messages.getLongPollHistory.
- `lp_version` (`int`, default: `3`) – Версия Long Poll.

`messages_is_messages_from_group_allowed`(*user_id*)

Параметры

`user_id` (`int`) – Идентификатор пользователя.

`messages_mark_as_answered_conversation`(*peer_id*, *answered=1*)

Параметры

- `peer_id` (`int`) – Идентификатор беседы.
- `answered` (`int`, default: `1`) – Беседа отмечена отвеченной.

`messages_mark_as_important_conversation`(*peer_id*, *important=1*)

Параметры

- `peer_id` (`int`) – Идентификатор беседы
- `important` (`int`, default: `1`) – Если сообщения необходимо пометить, как важные.

`messages_mark_as_read`(*message_ids=None*, *peer_id=None*, *start_message_id=None*, *mark_conversation_as_read=None*)

Параметры

- `message_ids` (`Optional[List[int]]`, default: `None`) – Идентификаторы сообщений.

- peer_id (Optional[str], default: None) – Идентификатор назначения.
- start_message_id (Optional[int], default: None) – При передаче этого параметра будут помечены как прочитанные все сообщения, начиная с данного.
- mark_conversation_as_read (Optional[int], default: None) –

messages_pin(*peer_id*, *message_id*, *conversation_message_id*)

Параметры

- peer_id (int) – Идентификатор назначения.
- message_id (int) – Идентификатор сообщения, которое нужно закрепить.
- conversation_message_id (int) – Идентификатор сообщения беседы, которое нужно закрепить.

messages_remove_chat_user(*chat_id*, *user_id=None*, *member_id=None*)

Параметры

- chat_id (int) – Идентификатор беседы.
- user_id (Optional[int], default: None) – Идентификатор пользователя, которого необходимо исключить из беседы.
- member_id (Optional[int], default: None) – Идентификатор участника, которого необходимо исключить из беседы.

Для сообществ — идентификатор сообщества со знаком «минус».

messages_restore(*message_id*)

Параметры

message_id (int) – Идентификатор сообщения, которое нужно восстановить.

messages_search(*q=None*, *peer_id=None*, *date=None*, *preview_length=0*, *offset=None*, *count=20*, *extended=None*, *fields=None*)

Параметры

- q (Optional[str], default: None) – Подстрока, по которой будет производиться поиск.
- peer_id (Optional[int], default: None) – Фильтр по идентификатору назначения для поиска по отдельному диалогу.
- date (Optional[int], default: None) – Дата в формате DDMMYYYY. Если параметр задан, в ответе будут только сообщения, отправленные до указанной даты.
- preview_length (int, default: 0) – Количество символов, по которому нужно обрезать сообщение.
- offset (Optional[int], default: None) – Смещение, необходимое для выборки определенного подмножества сообщений из списка найденных.
- count (int, default: 20) – Количество сообщений, которое необходимо получить.
- extended (Optional[int], default: None) – Возвращать дополнительные поля для пользователей и сообществ.

- fields (Optional[List[str]], default: None) – Список ыудополнительных полей для пользователей и сообществ.

messages_search_conversations(*q*, *count=20*, *extended=None*, *fields=None*)

Параметры

- q (str) – Поисковой запрос.
- count (int, default: 20) – Максимальное число результатов для получения.
- extended (Optional[int], default: None) – Возвращать дополнительные поля.
- fields (Optional[List[str]], default: None) – Дополнительные поля пользователей и сообществ, которые необходимо вернуть в ответе.

messages_send(*user_id=None*, *peer_id=None*, *peer_ids=None*, *domain=None*, *chat_id=None*, *message=None*, *lat=None*, *long=None*, *attachment=None*, *reply_to=None*, *forward_messages=None*, *forward=None*, *sticker_id=None*, *keyboard=None*, *template=None*, *payload=None*, *content_source=None*, *dont_parse_links=None*, *disable_mentions=None*, *intent='default'*, *subscribe_id=None*)

Параметры

- user_id (Optional[int], default: None) – Идентификатор пользователя, которому отправляется сообщение.
- peer_id (Optional[int], default: None) – Идентификатор назначения.
- peer_ids (Optional[List[int]], default: None) – Идентификаторы получателей сообщения (при необходимости отправить сообщение сразу нескольким пользователям)
- domain (Optional[str], default: None) – Короткий адрес пользователя.
- chat_id (Optional[int], default: None) – Идентификатор беседы, к которой будет относиться сообщение.
- message (Optional[str], default: None) – Текст личного сообщения. Обязательный параметр, если не задан параметр attachment.
- lat (Optional[float], default: None) – Географическая широта (от -90 до 90).
- long (Optional[float], default: None) – Географическая долгота (от -180 до 180).
- attachment (default: None) – Медиавложения к личному сообщению, перечисленные через запятую.
- reply_to (Optional[int], default: None) – Идентификатор сообщения, на которое требуется ответить.
- forward_messages (Optional[List[int]], default: None) – Идентификаторы пересылаемых сообщений.
- forward (default: None) – JSON-объект.
- sticker_id (Optional[int], default: None) – Идентификатор стикера.
- keyboard (default: None) – Объект, описывающий клавиатуру бота.
- template (Optional[Dict], default: None) – Объект, описывающий шаблон сообщения.
- payload (default: None) – Полезная нагрузка.

- content_source (Optional[Dict], default: None) – Объект, описывающий источник пользовательского контента для чат-ботов.
- dont_parse_links (Optional[int], default: None) – Не создавать сниппет ссылки из сообщения.
- disable_mentions (Optional[int], default: None) – Отключить уведомление об упоминании в сообщении.
- intent (str, default: 'default') – Строка, описывающая интенты.
- subscribe_id (Optional[int], default: None) –

messages_send_message_event_answer(*event_id*, *user_id*, *peer_id*, *event_data=None*)

Параметры

- event_id (str) – Случайная строка, которая возвращается в событии message_event.
- user_id (int) – Идентификатор пользователя.
- peer_id (int) – Идентификатор диалога со стороны сообщества.
- event_data (default: None) – Объект действия, которое должно произойти после нажатия на кнопку.

messages_set_activity(*user_id*, *type*, *peer_id*)

Параметры

- user_id (int) – Идентификатор пользователя.
- type (str) – typing — пользователь начал набирать текст, audiomessage — пользователь записывает голосовое сообщение.
- peer_id (int) – Идентификатор назначения.

messages_set_chat_photo(*file*)

Параметры

file (str) – Содержимое поля response из ответа специального upload сервера, полученного в результате загрузки изображения на адрес, полученный методом photos.getChatUploadServer.

messages_unpin(*peer_id*)

Параметры

peer_id (int) – Идентификатор назначения.

groups_add_address(*title*, *address*, *country_id*, *city_id*, *latitude*, *longitude*, *metro_id=None*, *additional_address=None*, *phone=None*, *work_info_status=None*, *timetable=None*, *is_main_address=None*)

Параметры

- title (str) – Заголовок адреса.
- address (str) – Строка адреса.
- additional_address (Optional[str], default: None) – Дополнительное описание адреса.
- country_id (int) – Идентификатор страны.
- city_id (int) – Идентификатор города.

- latitude (float) – Географическая широта отметки, заданная в градусах (от -90 до 90).
- longitude (float) – Географическая долгота отметки, заданная в градусах (от -180 до 180)
- metro_id (Optional[int], default: None) – Идентификатор станции метро.
- phone (Optional[str], default: None) – Номер телефона.
- work_info_status (Optional[str], default: None) – тип расписания. Возможные значения: no_information – нет информации о расписании; temporarily_closed — временно закрыто; always_opened — открыто круглосуточно; forever_closed — закрыто навсегда; timetable — открыто в указанные часы работы. Для этого типа расписания необходимо передать параметр timetable;
- timetable (default: None) –
- is_main_address (Optional[int], default: None) – Установить адрес основным.

groups_delete_address(*address_id*)

Параметры

address_id (int) – Идентификатор адреса.

groups_disable_online()

groups_edit(*title=None*, *description=None*, *screen_name=None*, *access=None*, *website=None*, *subject=None*, *email=None*, *phone=None*, *rss=None*, *event_start_date=None*, *event_finish_date=None*, *event_group_id=None*, *public_category=None*, *public_subcategory=None*, *public_date=None*, *wall=None*, *topics=None*, *photos=None*, *video=None*, *audio=None*, *links=None*, *events=None*, *places=None*, *contacts=None*, *docs=None*, *wiki=None*, *messages=None*, *articles=None*, *addresses=None*, *age_limits=None*, *market=None*, *market_comments=None*, *market_country=None*, *market_city=None*, *market_currency=None*, *market_contact=None*, *market_wiki=None*, *obscene_filter=None*, *obscene_stopwords=None*, *obscene_words=None*, *main_section=None*, *secondary_section=None*, *country=None*, *city=None*)

Параметры

- title (Optional[str], default: None) – Название сообщества.
- description (Optional[str], default: None) – Описание сообщества.
- screen_name (Optional[str], default: None) – Короткое имя сообщества.
- access (Optional[int], default: None) – тип группы. Возможные значения: 0 – открытая; 1 – закрытая; 2 – частная;
- website (Optional[str], default: None) – Адрес сайта, который будет указан в информации о группе.
- subject (Optional[int], default: None) – Тематика сообщества.
- email (Optional[str], default: None) – Электронный адрес организатора (для мероприятий).
- phone (Optional[str], default: None) – Номер телефона организатора (для мероприятий).
- rss (Optional[str], default: None) – Адрес rss для импорта новостей
- event_start_date (Optional[int], default: None) – Дата начала события.

- `event_finish_date` (`Optional[int]`, default: `None`) – Дата окончания события.
- `event_group_id` (`Optional[int]`, default: `None`) – идентификатор группы, которая является организатором события (только для событий).
- `public_category` (`Optional[int]`, default: `None`) – Категория публичной страницы.
- `public_subcategory` (`Optional[int]`, default: `None`) – Подкатегория публичной станицы.
- `public_date` (`Optional[str]`, default: `None`) – дата основания компании, организации, которой посвящена публичная страница в виде строки формата «dd.mm.YYYY».
- `wall` (`Optional[int]`, default: `None`) – Стена. Возможные значения: 0 — выключена; 1 — открытая; 2 — ограниченная (доступно только для групп и событий); 3 — закрытая (доступно только для групп и событий).
- `topics` (`Optional[int]`, default: `None`) – Обсуждения. Возможные значения: 0 — выключены; 1 — открытые; 2 — ограниченные (доступно только для групп и событий).
- `photos` (`Optional[int]`, default: `None`) – Фотографии. Возможные значения: 0 — выключены; 1 — открытые; 2 — ограниченные (доступно только для групп и событий).
- `video` (`Optional[int]`, default: `None`) – Видеозаписи. Возможные значения: 0 — выключены; 1 — открытые; 2 — ограниченные (доступно только для групп и событий).
- `audio` (`Optional[int]`, default: `None`) – Аудиозаписи. Возможные значения: 0 — выключены; 1 — открытые; 2 — ограниченные (доступно только для групп и событий).
- `links` (`Optional[int]`, default: `None`) – Ссылки (доступно только для публичных страниц). Возможные значения: 0 — выключены; 1 — включены.
- `events` (`Optional[int]`, default: `None`) – События (доступно только для публичных страниц). Возможные значения: 0 — выключены; 1 — включены.
- `places` (`Optional[int]`, default: `None`) – Места (доступно только для публичных страниц). Возможные значения: 0 — выключены; 1 — включены.
- `contacts` (`Optional[int]`, default: `None`) – Контакты (доступно только для публичных страниц). Возможные значения: 0 — выключены; 1 — включены.
- `docs` (`Optional[int]`, default: `None`) – Документы сообщества. Возможные значения: 0 — выключены; 1 — открытые; 2 — ограниченные (доступно только для групп и событий).
- `wiki` (`Optional[int]`, default: `None`) – wiki-материалы сообщества. Возможные значения: 0 — выключены; 1 — открытые; 2 — ограниченные (доступно только для групп и событий).
- `messages` (`Optional[int]`, default: `None`) – Сообщения сообщества. Возможные значения: 0 — выключены; 1 — включены.
- `articles` (`Optional[int]`, default: `None`) –
- `addresses` (`Optional[int]`, default: `None`) –

- age_limits (Optional[int], default: None) – Возрастное ограничение для сообщества. Возможные значения: 1 — нет ограничений; 2 — 16+; 3 — 18+.
- market (Optional[int], default: None) – Товары. Возможные значения: 0 — выключены; 1 — включены.
- market_comments (Optional[int], default: None) – Комментарии к товарам. Возможные значения: 0 — выключены; 1 — включены.
- market_country (Optional[List[int]], default: None) – Регионы доставки товаров.
- market_city (Optional[List[int]], default: None) – Города доставки товаров (в случае если указана одна страна).
- market_currency (Optional[int], default: None) – Идентификатор валюты магазина. Возможные значения: 643 — российский рубль; 980 — украинская гривна; 398 — казахстанский тенге; 978 — евро; 840 — доллар США.
- market_contact (Optional[int], default: None) – Контакт для связи для продавцом.
- market_wiki (Optional[int], default: None) – Идентификатор wiki-страницы с описанием магазина.
- obscene_filter (Optional[int], default: None) – Фильтр нецензурных выражений в комментариях. Возможные значения: 0 — выключен; 1 — включен.
- obscene_stopwords (Optional[int], default: None) – Фильтр по ключевым словам в комментариях. Возможные значения: 0 — выключен; 1 — включен.
- obscene_words (Optional[List[str]], default: None) – Ключевые слова для фильтра комментариев.
- main_section (Optional[int], default: None) –
- secondary_section (Optional[int], default: None) –
- country (Optional[int], default: None) –
- city (Optional[int], default: None) –

groups_edit_address(*address_id*, *title=None*, *address=None*, *additional_address=None*, *country_id=None*, *city_id=None*, *metro_id=None*, *latitude=None*, *longitude=None*, *phone=None*, *work_info_status=None*, *timetable=None*, *is_main_address=None*)

Параметры

- address_id (int) – Идентификатор адреса.
- title (Optional[str], default: None) – Заголовок адреса.
- address (Optional[str], default: None) – Строка адреса. *Невский проспект, дом 28*
- additional_address (Optional[str], default: None) – Дополнительное описание адреса.
- country_id (Optional[int], default: None) – Идентификатор страны.
- city_id (Optional[int], default: None) – Идентификатор города.
- metro_id (Optional[int], default: None) – Идентификатор станции метро.

- latitude (Optional[float], default: None) – Географическая широта отметки, заданная в градусах (от -90 до 90).
- longitude (Optional[float], default: None) – Географическая долгота отметки, заданная в градусах (от -180 до 180).
- phone (Optional[str], default: None) – Номер телефона.
- work_info_status (Optional[str], default: None) – Тип расписания.
- timetable (default: None) –
- is_main_address (Optional[int], default: None) – Установить адрес основным.

groups_enable_online()

groups_get_by_id(*fields=None*)

Параметры

fields (Optional[List[str]], default: None) – Список дополнительных полей, которые необходимо вернуть.

groups_get_members(*sort='id_asc'*, *offset=0*, *count=1000*, *fields=None*)

Параметры

- sort (str, default: 'id_asc') – Сортировка, с которой необходимо вернуть список участников. Возможные значения: id_asc — в порядке возрастания id; id_desc — в порядке убывания id; time_asc — в хронологическом порядке по вступлению в сообщество; time_desc — в антихронологическом порядке по вступлению в сообщество.
- offset (int, default: 0) – Смещение, необходимое для выборки определенного подмножества участников.
- count (int, default: 1000) – Количество участников сообщества, информацию о которых необходимо получить.
- fields (Optional[List[str]], default: None) – Список дополнительных полей, которые необходимо вернуть.

groups_is_member(*user_id=None*, *user_ids=None*)

Параметры

- user_id (Optional[int], default: None) – Идентификатор пользователя.
- user_ids (Optional[List[int]], default: None) – Идентификаторы пользователей, не более 500.

groups_get_banned(*fields=None*, *count=20*, *owner_id=None*, *offset=None*)

Параметры

- fields (Optional[List[str]], default: None) – Смещение, необходимое для выборки определенного подмножества черного списка.
- count (int, default: 20) – Количество пользователей, которое необходимо вернуть.
- owner_id (Optional[int], default: None) – Идентификатор пользователя или сообщества из чёрного списка, информацию о котором нужно получить.
- offset (Optional[int], default: None) – Смещение, необходимое для выборки определенного подмножества черного списка.

groups_get_online_status()

groups_get_token_permissions()

groups_set_settings(*messages=None*, *bots_capabilities=None*, *bots_start_button=None*, *bots_add_to_chat=None*)

Параметры

- messages (Optional[int], default: None) – Сообщения сообщества. Возможные значения: 0 — выключены; 1 — включены.
- bots_capabilities (Optional[int], default: None) – Возможности ботов (использование клавиатуры, добавление в беседу). Возможные значения: 0 — выключены; 1 — включены.
- bots_start_button (Optional[int], default: None) – Кнопка «Начать» в диалоге с сообществом. Работает, в случае если bots_capabilities=1. Возможные значения: 0 — выключена; 1 — включена.
- bots_add_to_chat (Optional[int], default: None) – Добавление бота в беседы. Работает, в случае если bots_capabilities=1. Возможные значения: 0 — запрещено; 1 — разрешено.

groups_get_longpoll_server()

board_delete_comment(*topic_id*, *comment_id*)

Параметры

- topic_id (int) – Идентификатор обсуждения.
- comment_id (int) – Идентификатор комментария в обсуждении.

board_restore_comment(*topic_id*, *comment_id*)

Параметры

- topic_id (int) – Идентификатор обсуждения.
- comment_id (int) – Идентификатор комментария.

docs_get_messages_upload_server(*peer_id*, *type='doc'*)

Параметры

- peer_id (int) – Идентификатор назначения.
- type (str, default: 'doc') – Тип документа. Возможные значения: doc — обычный документ; audio_message — голосовое сообщение;

docs_get_wall_upload_server()

docs_save(*file*, *title=None*, *tags=None*, *return_tags=None*)

Параметры

- file – Параметр, возвращаемый в результате загрузки файла на сервер.
- title (Optional[str], default: None) – Название документа.
- tags (Optional[List[str]], default: None) – Метки для поиска.
- return_tags (Optional[int], default: None) –

docs_search(*q*, *search_own=None*, *count=20*, *offset=None*, *return_tags=None*)

Параметры

- q (str) – Строка поискового запроса. Например, зеленые тапочки.
- search_own (Optional[int], default: None) – 1 — искать среди собственных документов пользователя.
- count (int, default: 20) – Количество документов, информацию о которых нужно вернуть.
- offset (Optional[int], default: None) – Смещение, необходимое для выборки определенного подмножества документов.
- return_tags (Optional[int], default: None) –

market_edit_order(*user_id*, *order_id*, *merchant_comment=None*, *status=None*, *track_number=None*, *payment_status=None*, *delivery_price=None*, *width=None*, *length=None*, *height=None*, *weight=None*, *comment_for_user=None*)

Параметры

- user_id (int) – Идентификатор пользователя.
- order_id (int) – Идентификатор заказа.
- merchant_comment (Optional[str], default: None) – Комментарий продавца.
- status (Optional[int], default: None) – Статус заказа. Возможные значения:

  0 - новый; 1 - согласуется; 2 - собирается; 3 - доставляется; 4 - выполнен; 5 - отменен; 6 - возвращен.
- track_number (Optional[str], default: None) – Трек-номер.
- payment_status (Optional[str], default: None) – Статус платежа. Возможные значения:

  not_paid - не оплачен; paid - оплачен; returned - возвращен.
- delivery_price (Optional[int], default: None) – Стоимость доставки.
- width (Optional[float], default: None) – Ширина.
- length (Optional[float], default: None) – Длина.
- height (Optional[float], default: None) – Высота.
- weight (Optional[float], default: None) – Вес.
- comment_for_user (Optional[str], default: None) –

market_get_group_orders(*offset=None*, *count=10*)

Параметры

- offset (Optional[int], default: None) – Смещение относительно первого найденного заказа для выборки определенного подмножества.
- count (int, default: 10) – Количество возвращаемых заказов.

market_get_order_by_id(*order_id*, *user_id=None*, *extended=None*)

Параметры

- order_id (int) – Идентификатор заказа.
- user_id (Optional[int], default: None) – Идентификатор пользователя.
- extended (Optional[int], default: None) –

market_get_order_items(*order_id*, *user_id=None*, *offset=None*, *count=50*)

Параметры

- order_id (int) – Идентификатор заказа.
- user_id (Optional[int], default: None) – Id пользователя, который сделал заказ.
- offset (Optional[int], default: None) – Смещение относительно первого найденного товара в заказе для выборки определенного подмножества.
- count (int, default: 50) – Количество возвращаемых товаров в заказе.

photos_get_chat_upload_server(*chat_id*, *crop_x=None*, *crop_y=None*, *crop_width=None*)

Параметры

- chat_id (int) – Идентификатор беседы, для которой нужно загрузить фотографию.
- crop_x (Optional[float], default: None) – Координата x для обрезки фотографии (верхний правый угол).
- crop_y (Optional[float], default: None) – Координата y для обрезки фотографии (верхний правый угол).
- crop_width (Optional[int], default: None) – Ширина фотографии после обрезки в px.

photos_get_messages_upload_server()

photos_get_owner_cover_photo_upload_server(*crop_x=None*, *crop_y=None*, *crop_x2=None*, *crop_y2=None*)

Параметры

- crop_x (Optional[float], default: None) – Координата X верхнего левого угла для обрезки изображения.
- crop_y (Optional[float], default: None) – Координата Y верхнего левого угла для обрезки изображения.
- crop_x2 (Optional[float], default: None) – Координата X нижнего правого угла для обрезки изображения.
- crop_y2 (Optional[float], default: None) – Координата Y нижнего правого угла для обрезки изображения.

photos_save_messages_photo(*photo*, *server=None*, *hash=None*)

Параметры

- photo (str) – Параметр, возвращаемый в результате загрузки фотографии на сервер.
- server (Optional[int], default: None) – Параметр, возвращаемый в результате загрузки фотографии на сервер.
- hash (Optional[str], default: None) – Параметр, возвращаемый в результате загрузки фотографии на сервер.

photos_save_owner_cover_photo(*hash*, *photo*)

Параметры

- **hash** (str) – Параметр hash, полученный в результате загрузки фотографии на сервер.
- **photo** (str) – Параметр photo, полученный в результате загрузки фотографии на сервер.

podcasts_search_podcast(*search_string*, *offset=None*, *count=20*)

Параметры

- **search_string** (str) –
- **offset** (Optional[int], default: None) –
- **count** (int, default: 20) –

storage_get(*key=None*, *keys=None*, *user_id=None*)

Параметры

- **key** (Optional[str], default: None) – Название переменной.
- **keys** (Optional[List[str]], default: None) – Список названий переменных. Если указано, параметр key не учитывается.
- **user_id** (Optional[int], default: None) – id пользователя, переменная которого устанавливается, в случае если данные запрашиваются серверным методом.

storage_get_keys(*user_id*, *offset=None*, *count=100*)

Параметры

- **user_id** (int) – id пользователя, названия переменных которого получаются, в случае если данные запрашиваются серверным методом.
- **offset** (Optional[int], default: None) – Смещение, необходимое для выборки определенного подмножества названий переменных
- **count** (int, default: 100) – Количество названий переменных, информацию о которых необходимо получить.

Тип результата
List

storage_set(*key*, *value=None*, *user_id=None*)

Параметры

- **key** (str) – Название переменной. Может содержать символы латинского алфавита, цифры, знак тире, нижнее подчёркивание [a-zA-Z_-0-9].
- **value** (Optional[str], default: None) – Значение переменной, сохраняются только первые 4096 байта.
- **user_id** (Optional[int], default: None) – id пользователя, переменная которого устанавливается, в случае если данные запрашиваются серверным методом.

users_get(*user_ids*, *fields=None*, *name_case='nom'*)

Параметры

- **user_ids** (List[int]) – Идентификаторы пользователей

- fields (Optional[List[str]], default: None) – Список дополнительных полей профилей, которые необходимо вернуть.
- name_case (str, default: 'nom') – Падеж для склонения имени и фамилии пользователя. Возможные значения:

  Именительный – nom, Родительный – gen, Дательный – dat, Винительный – acc, Творительный – ins, Предложный – abl.

stories_delete(*owner_id*, *story_id*, *stories=None*)

Параметры

- owner_id (int) – Идентификатор владельца истории.
- story_id (int) – Идентификатор истории.
- stories (Optional[List[str]], default: None) –

stories_get(*owner_id*, *extended=None*, *fields=None*)

Параметры

- owner_id (int) – Идентификатор пользователя, истории которого необходимо получить.
- extended (Optional[int], default: None) – 1 — возвращать в ответе дополнительную информацию о профилях пользователей.
- fields (Optional[List[str]], default: None) –

stories_get_by_id(*stories*, *extended=None*, *fields=None*)

Параметры

- stories (List[int]) – Идентификаторы историй.
- extended (Optional[int], default: None) – 1 — возвращать в ответе дополнительную информацию о пользователях.
- fields (Optional[List[str]], default: None) – Дополнительные поля профилей и сообществ, которые необходимо вернуть в ответе.

stories_get_photo_upload_server(*add_to_news=None*, *user_ids=None*, *reply_to_story=None*, *link_text=None*, *link_url=None*, *clickable_stickers=None*)

Параметры

- add_to_news (Optional[int], default: None) – 1 — разместить историю в новостях. Обязательно, если не указан user_ids.
- user_ids (Optional[List[int]], default: None) – Идентификаторы пользователей, которые будут видеть историю (для отправки в личном сообщении). Обязательно, если add_to_news не передан.
- reply_to_story (Optional[str], default: None) – Идентификатор истории, в ответ на которую создается новая.
- link_text (Optional[str], default: None) – Текст ссылки для перехода из истории.
- link_url (Optional[str], default: None) – Адрес ссылки для перехода из истории. Допустимы только внутренние ссылки https://vk.com.
- clickable_stickers (default: None) – Объект кликабельного стикера.

stories_get_replies(*owner_id*, *story_id*, *access_key=None*, *extended=None*, *fields=None*)

Параметры

- owner_id (int) – Идентификатор владельца истории.
- story_id (int) – Идентификатор истории.
- access_key (Optional[str], default: None) – Ключ доступа для приватного объекта.
- extended (Optional[int], default: None) – 1 — возвращать дополнительную информацию о профилях и сообществах.
- fields (Optional[List[str]], default: None) – Дополнительные поля профилей и сообществ, которые необходимо вернуть в ответе.

stories_get_stats(*owner_id*, *story_id*)

Параметры

- owner_id (int) – Идентификатор владельца истории.
- story_id (int) – Идентификатор истории.

stories_get_video_upload_server(*user_ids=None*, *add_to_news=None*, *reply_to_story=None*, *link_text=None*, *link_url=None*, *clickable_stickers=None*)

Параметры

- user_ids (Optional[List[int]], default: None) – Идентификаторы пользователей, которые будут видеть историю (для отправки в личном сообщении).
- add_to_news (Optional[int], default: None) – 1 — разместить историю в новостях.
- reply_to_story (Optional[str], default: None) – Идентификатор истории, в ответ на которую создается новая.
- link_text (Optional[str], default: None) – Текст ссылки для перехода из истории.
- link_url (Optional[str], default: None) – Адрес ссылки для перехода из истории.
- clickable_stickers (default: None) – Объект кликабельного стикера.

stories_get_viewers(*owner_id*, *story_id*, *count=100*, *offset=None*, *extended=None*)

Параметры

- owner_id (int) – Идентификатор владельца истории.
- story_id (int) – Идентификатор истории.
- count (int, default: 100) – Максимальное число результатов в ответе.
- offset (Optional[int], default: None) – Сдвиг для получения определённого подмножества результатов.
- extended (Optional[int], default: None) – 1 — возвращать в ответе расширенную информацию о пользователях.

stories_hide_all_replies(*owner_id*)

Параметры
owner_id (int) – Идентификатор пользователя, ответы от которого нужно скрыть.

stories_hide_reply(*owner_id*, *story_id*)

Параметры

- owner_id (int) – Идентификатор владельца истории (ответной).
- story_id (int) – Идентификатор истории (ответной).

stories_save(*upload_results*, *extended=None*, *fields=None*)

Параметры

- upload_results (List[str]) – Список строк, которые возвращает stories.getPhotoUploadServer или stories.getVideoUploadServer.
- extended (Optional[int], default: None) –
- fields (Optional[List[str]], default: None) –

utils_check_link(*url*)

Параметры
url (str) – Внешняя ссылка, которую необходимо проверить.

utils_get_link_stats(*key*, *source='vk_cc'*, *access_key=None*, *interval='day'*, *intervals_count=1*, *extended=None*)

Параметры

- key (str) – Сокращенная ссылка (часть URL после «vk.cc/»).
- source (str, default: 'vk_cc') –
- access_key (Optional[str], default: None) – Ключ доступа к приватной статистике ссылки.
- interval (str, default: 'day') – Единица времени для подсчета статистики. Возможные значения: hour, day, week, month, forever.
- intervals_count (int, default: 1) – Длительность периода для получения статистики в выбранных единицах (из параметра interval).
- extended (Optional[int], default: None) – 1 — возвращать расширенную статистику (пол/возраст/страна/город), 0 — возвращать только количество переходов.

utils_get_server_time()

Возвращает число, соответствующее времени в UnixTime.

utils_get_short_link(*url*, *private=None*)

Параметры

- url (str) – URL, для которого необходимо получить сокращенный вариант.
- private (Optional[int], default: None) – 1 — статистика ссылки приватная, 0 — статистика ссылки общедоступная.

utils_resolve_screen_name(*screen_name*)

Параметры

screen_name (str) – Короткое имя пользователя, группы или приложения.

wall_close_comments(*owner_id*, *post_id*)

Параметры

- owner_id (int) –
- post_id (int) –

wall_create_comment(*post_id*, *message=None*, *reply_to_comment=None*, *attachments=None*, *sticker_id=None*)

Параметры

- post_id (int) – Идентификатор записи на стене.
- message (Optional[str], default: None) – Текст комментария. Обязательный параметр, если не передан параметр attachments.
- reply_to_comment (Optional[int], default: None) – Идентификатор комментария, в ответ на который должен быть добавлен новый комментарий.
- attachments (default: None) –
- sticker_id (Optional[int], default: None) – Идентификатор стикера.

vk_maria.dispatcher

vk_maria.dispatcher.dispatcher
vk_maria.dispatcher.filters
vk_maria.dispatcher.fsm

vk_maria.dispatcher.dispatcher

Classes

Dispatcher	**param storage**

vk_maria.dispatcher.dispatcher.Dispatcher

class Dispatcher(*vk*, *storage=<vk_maria.dispatcher.fsm.storage.core.DisabledStorage object>*)

Базовые классы: object

Параметры

storage (Optional[*BaseStorage*], default: <vk_maria.dispatcher.fsm.storage.core.DisabledStorage object at 0x7f488ad8c4f0>) –

Methods

callback_handler	
event_handler	**param event_type**
message_handler	**param commands**
register_callback_handler	**param function**
register_event_handler	**param function**
register_message_handler	**param function**
start_polling	

register_event_handler(*function*, *event_type*, **filters*, ***bound_filters*)

Параметры

- function (callable) –
- event_type (*EventType*) –

register_message_handler(*function*, **filters*, *commands=None*, *frm='user'*, *regexp=None*, *state=None*, ***kwargs*)

Параметры

- function (callable) –
- commands (Optional[List[str]], default: None) –
- frm (str, default: 'user') –
- regexp (Optional[str], default: None) –

register_callback_handler(*function*, **filters*, *state=None*, ***bound_filters*)

Параметры

function (callable) –

event_handler(*event_type*, **filters*, ***bound_filters*)

Параметры
event_type (*EventType*) –

message_handler(**filters*, *commands=None*, *frm='user'*, *regexp=None*, *state=None*, ***bound_filters*)

Параметры

- commands (Optional[List[str]], default: None) –
- frm (str, default: 'user') –
- regexp (Optional[str], default: None) –

callback_handler(**filters*, *state=None*, *payload=None*, ***bound_filters*)

start_polling(*debug=False*, *on_startup=None*, *on_shutdown=None*)

class Dispatcher(*vk*, *storage=<vk_maria.dispatcher.fsm.storage.core.DisabledStorage object>*)

Параметры
storage (Optional[*BaseStorage*], default: <vk_maria.dispatcher.fsm.storage.core.DisabledStorage object at 0x7f488ad8c4f0>) –

register_event_handler(*function*, *event_type*, **filters*, ***bound_filters*)

Параметры

- function (callable) –
- event_type (*EventType*) –

register_message_handler(*function*, **filters*, *commands=None*, *frm='user'*, *regexp=None*, *state=None*, ***kwargs*)

Параметры

- function (callable) –
- commands (Optional[List[str]], default: None) –
- frm (str, default: 'user') –
- regexp (Optional[str], default: None) –

register_callback_handler(*function*, **filters*, *state=None*, ***bound_filters*)

Параметры
function (callable) –

event_handler(*event_type*, **filters*, ***bound_filters*)

Параметры
event_type (*EventType*) –

message_handler(**filters*, *commands=None*, *frm='user'*, *regexp=None*, *state=None*, ***bound_filters*)

Параметры

- commands (Optional[List[str]], default: None) –
- frm (str, default: 'user') –

- regexp (Optional[str], default: None) –

callback_handler(**filters*, *state=None*, *payload=None*, ***bound_filters*)

start_polling(*debug=False*, *on_startup=None*, *on_shutdown=None*)

vk_maria.dispatcher.filters

vk_maria.dispatcher.filters.filters
vk_maria.dispatcher.filters.handler

vk_maria.dispatcher.filters.filters

Classes

AbstractFilter	
BoundFilter	
BoundFilterMeta	
CommandsFilter	**param commands**
EventTypeFilter	**param event_type**
FSMStateFilter	**param state**
Filters	**param filters**
FiltersFactory	
FunctionFilter	
PayloadFilter	
RegexpFilter	**param regexp**
TextFilter	Можно использовать что-то одно из equals, contains, startswith и endswith
TypeFromFilter	**param frm**

vk_maria.dispatcher.filters.filters.AbstractFilter

class AbstractFilter

Базовые классы: ABC

Methods

check	param event

abstract check(*event*)

Параметры
event (*Event*) –

vk_maria.dispatcher.filters.filters.BoundFilter

class BoundFilter

Базовые классы: object

Methods

check	param event

Attributes

key

key: str

abstract check(*event*)

Параметры
event (*Event*) –

vk_maria.dispatcher.filters.filters.BoundFilterMeta

class BoundFilterMeta(*name*, *bases*, *namespace*)

Базовые классы: type

Methods

mro	Return a type's method resolution order.

Attributes

registered_filters

registered_filters = [<class 'vk_maria.dispatcher.filters.filters.EventTypeFilter'>, <class 'vk_maria.dispatcher.filters.filters.TextFilter'>, <class 'vk_maria.dispatcher.filters.filters.FunctionFilter'>, <class 'vk_maria.dispatcher.filters.filters.CommandsFilter'>, <class 'vk_maria.dispatcher.filters.filters.TypeFromFilter'>, <class 'vk_maria.dispatcher.filters.filters.RegexpFilter'>, <class 'vk_maria.dispatcher.filters.filters.FSMStateFilter'>, <class 'vk_maria.dispatcher.filters.filters.PayloadFilter'>]

__call__(**args*, ***kwargs*)

Call self as a function.

mro()

Return a type's method resolution order.

vk_maria.dispatcher.filters.filters.CommandsFilter

class CommandsFilter(*commands*, *prefixes=''*, *ignore_case=False*)

Базовые классы: *BoundFilter*

Параметры

- commands (List[str]) –
- prefixes (str, default: '') –
- ignore_case (bool, default: False) –

Methods

check	param event

Attributes

key

key: str = 'commands'

check(*event*)

Параметры
event (*MessageEvent*) –

vk_maria.dispatcher.filters.filters.EventTypeFilter

class EventTypeFilter(*event_type*)

Базовые классы: *BoundFilter*

Параметры
event_type (*EventType*) –

Methods

check	param event

Attributes

key

key: str = 'event_type'

check(*event*)

Параметры
event (*Event*) –

vk_maria.dispatcher.filters.filters.FSMStateFilter

class FSMStateFilter(*state*)

Базовые классы: *BoundFilter*

Параметры
state (str) –

Methods

check	param event

Attributes

key

key: str = 'state'

check(*event*)

Параметры
event (*MessageEvent*) –

vk_maria.dispatcher.filters.filters.Filters

class Filters(**filters*)

Базовые классы: object

Параметры
filters (*AbstractFilter*) –

Methods

check_all	param event

check_all(*event*)

Параметры
event (*Event*) –

Тип результата
bool

vk_maria.dispatcher.filters.filters.FiltersFactory

class FiltersFactory

Базовые классы: object

Methods

get_filter_by_key	**param key**
get_filters	

exception UnknownFilterException

Базовые классы: Exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

classmethod get_filters(**custom_filters*, ***bound_filters*)

classmethod get_filter_by_key(*key*, *filter_value*)

Параметры

key (str) –

vk_maria.dispatcher.filters.filters.FunctionFilter

class FunctionFilter(*f*)

Базовые классы: *BoundFilter*

Methods

check	**param event**

Attributes

key

key: str = 'f'

check(*event*)

Параметры
event (*MessageEvent*) –

vk_maria.dispatcher.filters.filters.PayloadFilter

class PayloadFilter(*payload*)

Базовые классы: *BoundFilter*

Methods

check	**param event**

Attributes

key

key: str = 'payload'

check(*event*)

Параметры
event (*CallbackQueryEvent*) –

vk_maria.dispatcher.filters.filters.RegexpFilter

class RegexpFilter(*regexp*)

Базовые классы: *BoundFilter*

Параметры
regexp (str) –

Methods

check	**param event**

Attributes

key

key: str = 'regexp'

check(*event*)

Параметры
event (*MessageEvent*) –

vk_maria.dispatcher.filters.filters.TextFilter

class TextFilter(*equals=None*, *contains=None*, *startswith=None*, *endswith=None*, *ignore_case=False*)

Базовые классы: *BoundFilter*

Можно использовать что-то одно из equals, contains, startswith и endswith

Параметры

- equals (Optional[str], default: None) –
- contains (Optional[str], default: None) –
- startswith (Optional[str], default: None) –
- endswith (Optional[str], default: None) –

Methods

check	**param event**

Attributes

key

key: str = 'text'

check(*event*)

Параметры
event (*MessageEvent*) –

vk_maria.dispatcher.filters.filters.TypeFromFilter

class TypeFromFilter(*frm*)

Базовые классы: *BoundFilter*

Параметры
frm (str) –

Methods

check	**param event**

Attributes

key

key: str = 'frm'

check(*event*)

Параметры
event (*MessageEvent*) –

class AbstractFilter

abstract check(*event*)

Параметры
event (*Event*) –

class BoundFilterMeta(*name*, *bases*, *namespace*)

registered_filters = [<class 'vk_maria.dispatcher.filters.filters.EventTypeFilter'>, <class 'vk_maria.dispatcher.filters.filters.TextFilter'>, <class 'vk_maria.dispatcher.filters.filters.FunctionFilter'>, <class 'vk_maria.dispatcher.filters.filters.CommandsFilter'>, <class 'vk_maria.dispatcher.filters.filters.TypeFromFilter'>, <class 'vk_maria.dispatcher.filters.filters.RegexpFilter'>, <class 'vk_maria.dispatcher.filters.filters.FSMStateFilter'>, <class 'vk_maria.dispatcher.filters.filters.PayloadFilter'>]

mro()

Return a type's method resolution order.

class BoundFilter

key: str

abstract check(*event*)

Параметры
event (*Event*) –

class EventTypeFilter(*event_type*)

Параметры
event_type (*EventType*) –

key: str = 'event_type'

check(*event*)

Параметры
event (*Event*) –

class TextFilter(*equals=None*, *contains=None*, *startswith=None*, *endswith=None*, *ignore_case=False*)

Можно использовать что-то одно из equals, contains, startswith и endswith

Параметры

- equals (Optional[str], default: None) –
- contains (Optional[str], default: None) –
- startswith (Optional[str], default: None) –
- endswith (Optional[str], default: None) –

key: str = 'text'

check(*event*)

Параметры
event (*MessageEvent*) –

class FunctionFilter(*f*)

key: str = 'f'

check(*event*)

Параметры
event (*MessageEvent*) –

class CommandsFilter(*commands*, *prefixes=''*, *ignore_case=False*)

Параметры

- commands (List[str]) –
- prefixes (str, default: '') –
- ignore_case (bool, default: False) –

key: str = 'commands'

check(*event*)

Параметры
event (*MessageEvent*) –

class TypeFromFilter(*frm*)

Параметры
frm (str) –

key: str = 'frm'

check(*event*)

Параметры
event (*MessageEvent*) –

class RegexpFilter(*regexp*)

Параметры
regexp (str) –

key: str = 'regexp'

check(*event*)

Параметры
event (*MessageEvent*) –

class FSMStateFilter(*state*)

Параметры
state (str) –

key: str = 'state'

check(*event*)

Параметры
event (*MessageEvent*) –

class PayloadFilter(*payload*)

key: str = 'payload'

check(*event*)

Параметры
event (*CallbackQueryEvent*) –

class Filters(**filters*)

Параметры
filters (*AbstractFilter*) –

check_all(*event*)

Параметры
event (*Event*) –

Тип результата
bool

class FiltersFactory

exception UnknownFilterException

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

classmethod get_filters(*custom_filters, **bound_filters)

classmethod get_filter_by_key(key, filter_value)

Параметры

key (str) –

vk_maria.dispatcher.filters.handler

Classes

HandlerManager	
HandlerObject	**param function**

vk_maria.dispatcher.filters.handler.HandlerManager

class HandlerManager

Базовые классы: object

Methods

register_handler	**param function**

register_handler(*function*, **filters*, ***bound_filters*)

Параметры

function (callable) –

vk_maria.dispatcher.filters.handler.HandlerObject

class HandlerObject(*function*, **filters*)

Базовые классы: object

Параметры

- function (callable) –
- filters (*AbstractFilter*) –

Methods

test_handler	param event

test_handler(*event*)

Параметры
event (*Event*) –

Тип результата
bool

__call__(**args*)

Call self as a function.

class HandlerObject(*function*, **filters*)

Параметры

- function (callable) –
- filters (*AbstractFilter*) –

test_handler(*event*)

Параметры
event (*Event*) –

Тип результата
bool

class HandlerManager

register_handler(*function*, **filters*, ***bound_filters*)

Параметры
function (callable) –

vk_maria.dispatcher.fsm

vk_maria.dispatcher.fsm.state
vk_maria.dispatcher.fsm.storage

vk_maria.dispatcher.fsm.state

Classes

State
StatesGroup
StatesGroupMeta

vk_maria.dispatcher.fsm.state.State

class State(*state*)

Базовые классы: object

Methods

set

Attributes

state

property state: str

set()

vk_maria.dispatcher.fsm.state.StatesGroup

class StatesGroup

Базовые классы: object

Methods

finish	
first	**rtype** str
last	**rtype** str
next	
previous	**rtype** str

classmethod next()

classmethod previous()

Тип результата
str

classmethod first()

Тип результата
str

classmethod last()

Тип результата
str

classmethod finish()

vk_maria.dispatcher.fsm.state.StatesGroupMeta

class StatesGroupMeta(*name*, *bases*, *namespace*, **args*, ***kwargs*)

Базовые классы: type

Methods

mro	Return a type's method resolution order.

__call__(**args*, ***kwargs*)

Call self as a function.

mro()

Return a type's method resolution order.

class State(*state*)

property state: str

set()

class StatesGroup

classmethod next()

classmethod previous()

Тип результата
str

classmethod first()

Тип результата
str

classmethod last()

Тип результата
str

classmethod finish()

vk_maria.dispatcher.fsm.storage

vk_maria.dispatcher.fsm.storage.core
vk_maria.dispatcher.fsm.storage.file
vk_maria.dispatcher.fsm.storage.memory

vk_maria.dispatcher.fsm.storage.core

Classes

BaseStorage	
DisabledStorage	
FSMContext	**param storage**

vk_maria.dispatcher.fsm.storage.core.BaseStorage

class BaseStorage

Базовые классы: ABC

Methods

check_address	**param chat**
close	
finish	**param chat**
get_data	**param chat**
get_state	**param chat**
reset_data	**param chat**
reset_state	**param chat**
set_data	**param chat**
set_state	**param chat**
update_data	**param chat**

abstract close()

classmethod check_address(*, *chat=None*, *user=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

Тип результата
(Union[str, int], Union[str, int])

abstract get_state(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –

- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Optional[str]

abstract get_data(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Dict

abstract set_state(*, *chat=None*, *user=None*, *state=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- state (Optional[AnyStr], default: None) –

abstract set_data(*, *chat=None*, *user=None*, *data=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

abstract update_data(*, *chat=None*, *user=None*, *data=None*, ***kwargs*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

reset_data(*, *chat=None*, *user=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

reset_state(*, *chat=None*, *user=None*, *with_ data=True*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- with_data (Optional[bool], default: True) –

finish(*, *chat=None*, *user=None*, *with_data=False*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- with_data (bool, default: False) –

vk_maria.dispatcher.fsm.storage.core.DisabledStorage

class DisabledStorage

Базовые классы: *BaseStorage*

Methods

check_address	**param chat**
close	
finish	**param chat**
get_data	**param chat**
get_state	**param chat**
reset_data	**param chat**
reset_state	**param chat**
set_data	**param chat**
set_state	**param chat**
update_data	**param chat**

close()

get_state(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Optional[str]

get_data(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Dict

update_data(*, *chat=None*, *user=None*, *data=None*, ***kwargs*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

set_state(*, *chat=None*, *user=None*, *state=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- state (Optional[AnyStr], default: None) –

set_data(*, *chat=None*, *user=None*, *data=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

classmethod check_address(*, *chat=None*, *user=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

Тип результата
(Union[str, int], Union[str, int])

finish(*, *chat=None*, *user=None*, *with_data=False*)

Параметры

- chat (Union[str, int, None], default: None) –

- **user** (Union[str, int, None], default: None) –
- **with_data** (bool, default: False) –

reset_data(*, *chat=None*, *user=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –

reset_state(*, *chat=None*, *user=None*, *with_data=True*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **with_data** (Optional[bool], default: True) –

vk_maria.dispatcher.fsm.storage.core.FSMContext

class FSMContext(*storage*)

Базовые классы: *Singleton*

Параметры

storage (*BaseStorage*) –

Methods

finish	**param with_data**
get_data	**param default**
get_state	**param default**
reset_data	
reset_state	**param with_data**
set_data	**param data**
set_state	**param state**
update_data	**param data**

get_state(*default=None*)

Параметры
default (Optional[str], default: None) –

Тип результата
Optional[str]

get_data(*default=None*)

Параметры
default (Optional[str], default: None) –

Тип результата
Dict

update_data(*data=None*, ***kwargs*)

Параметры
data (Optional[Dict], default: None) –

set_state(*state=None*)

Параметры
state (Optional[AnyStr], default: None) –

set_data(*data=None*)

Параметры
data (Optional[Dict], default: None) –

reset_state(*with_data=True*)

Параметры
with_data (Optional[bool], default: True) –

reset_data()

finish(*with_data=False*)

Параметры
with_data (bool, default: False) –

class BaseStorage

abstract close()

classmethod check_address(***, *chat=None*, *user=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

Тип результата
(Union[str, int], Union[str, int])

abstract get_state(***, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

- default (Optional[str], default: None) –

Тип результата
Optional[str]

abstract get_data(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Dict

abstract set_state(*, *chat=None*, *user=None*, *state=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- state (Optional[AnyStr], default: None) –

abstract set_data(*, *chat=None*, *user=None*, *data=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

abstract update_data(*, *chat=None*, *user=None*, *data=None*, ***kwargs*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

reset_data(*, *chat=None*, *user=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

reset_state(*, *chat=None*, *user=None*, *with_data=True*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- with_data (Optional[bool], default: True) –

finish(*, chat=None, user=None, with_data=False)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- with_data (bool, default: False) –

class FSMContext(storage)

Параметры

storage (*BaseStorage*) –

get_state(default=None)

Параметры

default (Optional[str], default: None) –

Тип результата

Optional[str]

get_data(default=None)

Параметры

default (Optional[str], default: None) –

Тип результата

Dict

update_data(data=None, **kwargs)

Параметры

data (Optional[Dict], default: None) –

set_state(state=None)

Параметры

state (Optional[AnyStr], default: None) –

set_data(data=None)

Параметры

data (Optional[Dict], default: None) –

reset_state(with_data=True)

Параметры

with_data (Optional[bool], default: True) –

reset_data()

finish(with_data=False)

Параметры

with_data (bool, default: False) –

class DisabledStorage

close()

get_state(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Optional[str]

get_data(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Dict

update_data(*, *chat=None*, *user=None*, *data=None*, ***kwargs*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

set_state(*, *chat=None*, *user=None*, *state=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- state (Optional[AnyStr], default: None) –

set_data(*, *chat=None*, *user=None*, *data=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

classmethod check_address(*, *chat=None*, *user=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

Тип результата
(Union[str, int], Union[str, int])

finish(*, *chat=None*, *user=None*, *with_data=False*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- with_data (bool, default: False) –

reset_data(*, *chat=None*, *user=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

reset_state(*, *chat=None*, *user=None*, *with_data=True*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- with_data (Optional[bool], default: True) –

vk_maria.dispatcher.fsm.storage.file

vk_maria.dispatcher.fsm.storage.file.base
vk_maria.dispatcher.fsm.storage.file.json
vk_maria.dispatcher.fsm.storage.file.pickle

vk_maria.dispatcher.fsm.storage.file.base

Classes

FileStorage	**param path**

vk_maria.dispatcher.fsm.storage.file.base.FileStorage

class FileStorage(*path*)

Базовые классы: *MemoryStorage*

Параметры

path (Union[Path, str]) –

Methods

check_address	**param chat**
close	
finish	**param chat**
get_data	**param chat**
get_state	**param chat**
read	**param path**
reset_data	**param chat**
reset_state	**param chat**
resolve_address	
set_data	**param chat**
set_state	**param chat**
update_data	**param chat**
write	**param path**

close()

read(*path*)

Параметры
path (Path) –

write(*path*)

Параметры
path (Path) –

classmethod check_address(*, *chat=None*, *user=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

Тип результата
(Union[str, int], Union[str, int])

finish(*, *chat=None*, *user=None*, *with_data=False*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- with_data (bool, default: False) –

get_data(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Dict

get_state(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Optional[str]

reset_data(*, *chat=None*, *user=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

reset_state(*, *chat=None*, *user=None*, *with_data=True*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- with_data (Optional[bool], default: True) –

resolve_address(*chat*, *user*)

set_data(*, *chat=None*, *user=None*, *data=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

set_state(*, *chat=None*, *user=None*, *state=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- state (Optional[AnyStr], default: None) –

update_data(*, *chat=None*, *user=None*, *data=None*, ***kwargs*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

class FileStorage(*path*)

Параметры

path (Union[Path, str]) –

close()

read(*path*)

Параметры

path (Path) –

write(*path*)

Параметры

path (Path) –

classmethod check_address(*, *chat=None*, *user=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

Тип результата
(Union[str, int], Union[str, int])

finish(*, *chat=None*, *user=None*, *with_data=False*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- with_data (bool, default: False) –

get_data(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Dict

get_state(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Optional[str]

reset_data(*, *chat=None*, *user=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

reset_state(*, *chat=None*, *user=None*, *with_data=True*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- with_data (Optional[bool], default: True) –

resolve_address(*chat*, *user*)

set_data(*, *chat=None*, *user=None*, *data=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

set_state(*, *chat=None*, *user=None*, *state=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- state (Optional[AnyStr], default: None) –

update_data(*, *chat=None*, *user=None*, *data=None*, ***kwargs*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

vk_maria.dispatcher.fsm.storage.file.json

Classes

JSONStorage	JSON File storage based on MemoryStorage

vk_maria.dispatcher.fsm.storage.file.json.JSONStorage

class JSONStorage(*path*)

Базовые классы: *FileStorage*

JSON File storage based on MemoryStorage

Параметры

path (Union[Path, str]) –

Methods

check_address	**param chat**
close	
finish	**param chat**
get_data	**param chat**
get_state	**param chat**
read	**param path**
reset_data	**param chat**
reset_state	**param chat**
resolve_address	
set_data	**param chat**
set_state	**param chat**
update_data	**param chat**
write	**param path**

`read`(*path*)

Параметры
`path` (`Path`) –

`write`(*path*)

Параметры
`path` (`Path`) –

`classmethod check_address`(***, *chat=None*, *user=None*)

Параметры

- `chat` (`Union[str, int, None]`, default: `None`) –

- **user** (Union[str, int, None], default: None) –

Тип результата
(Union[str, int], Union[str, int])

close()

finish(*, *chat=None*, *user=None*, *with_data=False*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **with_data** (bool, default: False) –

get_data(*, *chat=None*, *user=None*, *default=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **default** (Optional[str], default: None) –

Тип результата
Dict

get_state(*, *chat=None*, *user=None*, *default=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **default** (Optional[str], default: None) –

Тип результата
Optional[str]

reset_data(*, *chat=None*, *user=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –

reset_state(*, *chat=None*, *user=None*, *with_data=True*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **with_data** (Optional[bool], default: True) –

resolve_address(*chat*, *user*)

set_data(*, *chat=None*, *user=None*, *data=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **data** (Optional[Dict], default: None) –

set_state(*, *chat=None*, *user=None*, *state=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **state** (Optional[AnyStr], default: None) –

update_data(*, *chat=None*, *user=None*, *data=None*, ***kwargs*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **data** (Optional[Dict], default: None) –

class JSONStorage(*path*)

JSON File storage based on MemoryStorage

Параметры

path (Union[Path, str]) –

read(*path*)

Параметры

path (Path) –

write(*path*)

Параметры

path (Path) –

classmethod check_address(*, *chat=None*, *user=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –

Тип результата

(Union[str, int], Union[str, int])

close()

finish(*, *chat=None*, *user=None*, *with_data=False*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –

- **with_data** (bool, default: False) –

get_data(*, *chat=None*, *user=None*, *default=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **default** (Optional[str], default: None) –

Тип результата
Dict

get_state(*, *chat=None*, *user=None*, *default=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **default** (Optional[str], default: None) –

Тип результата
Optional[str]

reset_data(*, *chat=None*, *user=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –

reset_state(*, *chat=None*, *user=None*, *with_data=True*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **with_data** (Optional[bool], default: True) –

resolve_address(*chat*, *user*)

set_data(*, *chat=None*, *user=None*, *data=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **data** (Optional[Dict], default: None) –

set_state(*, *chat=None*, *user=None*, *state=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **state** (Optional[AnyStr], default: None) –

update_data(*, *chat=None*, *user=None*, *data=None*, ***kwargs*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

vk_maria.dispatcher.fsm.storage.file.pickle

Classes

PickleStorage	Pickle File storage based on MemoryStorage

vk_maria.dispatcher.fsm.storage.file.pickle.PickleStorage

class PickleStorage(*path*)

Базовые классы: *FileStorage*

Pickle File storage based on MemoryStorage

Параметры

path (Union[Path, str]) –

Methods

check_address	**param chat**
close	
finish	**param chat**
get_data	**param chat**
get_state	**param chat**
read	**param path**
reset_data	**param chat**
reset_state	**param chat**
resolve_address	
set_data	**param chat**
set_state	**param chat**
update_data	**param chat**
write	**param path**

`read`(*path*)

Параметры
`path` (`Path`) –

`write`(*path*)

Параметры
`path` (`Path`) –

`classmethod check_address`(*, *chat=None*, *user=None*)

Параметры

- `chat` (`Union[str, int, None]`, default: `None`) –

- **user** (Union[str, int, None], default: None) –

Тип результата
(Union[str, int], Union[str, int])

close()

finish(*, *chat=None*, *user=None*, *with_data=False*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **with_data** (bool, default: False) –

get_data(*, *chat=None*, *user=None*, *default=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **default** (Optional[str], default: None) –

Тип результата
Dict

get_state(*, *chat=None*, *user=None*, *default=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **default** (Optional[str], default: None) –

Тип результата
Optional[str]

reset_data(*, *chat=None*, *user=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –

reset_state(*, *chat=None*, *user=None*, *with_data=True*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **with_data** (Optional[bool], default: True) –

resolve_address(*chat*, *user*)

set_data(*, *chat=None*, *user=None*, *data=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

set_state(*, *chat=None*, *user=None*, *state=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- state (Optional[AnyStr], default: None) –

update_data(*, *chat=None*, *user=None*, *data=None*, ***kwargs*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

class PickleStorage(*path*)

Pickle File storage based on MemoryStorage

Параметры

path (Union[Path, str]) –

read(*path*)

Параметры

path (Path) –

write(*path*)

Параметры

path (Path) –

classmethod check_address(*, *chat=None*, *user=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

Тип результата

(Union[str, int], Union[str, int])

close()

finish(*, *chat=None*, *user=None*, *with_data=False*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

- **with_data** (bool, default: False) –

get_data(*, *chat=None*, *user=None*, *default=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **default** (Optional[str], default: None) –

Тип результата
Dict

get_state(*, *chat=None*, *user=None*, *default=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **default** (Optional[str], default: None) –

Тип результата
Optional[str]

reset_data(*, *chat=None*, *user=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –

reset_state(*, *chat=None*, *user=None*, *with_data=True*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **with_data** (Optional[bool], default: True) –

resolve_address(*chat*, *user*)

set_data(*, *chat=None*, *user=None*, *data=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **data** (Optional[Dict], default: None) –

set_state(*, *chat=None*, *user=None*, *state=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **state** (Optional[AnyStr], default: None) –

update_data(*, *chat=None*, *user=None*, *data=None*, ***kwargs*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

vk_maria.dispatcher.fsm.storage.memory

vk_maria.dispatcher.fsm.storage.memory.memory

vk_maria.dispatcher.fsm.storage.memory.memory

Classes

MemoryStorage

vk_maria.dispatcher.fsm.storage.memory.memory.MemoryStorage

class MemoryStorage

Базовые классы: *BaseStorage*

Methods

check_address	**param chat**
close	
finish	**param chat**
get_data	**param chat**
get_state	**param chat**
reset_data	**param chat**
reset_state	**param chat**
resolve_address	
set_data	**param chat**
set_state	**param chat**
update_data	**param chat**

close()

resolve_address(*chat*, *user*)

get_state(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Optional[str]

get_data(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –

- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Dict

set_state(*, *chat=None*, *user=None*, *state=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- state (Optional[AnyStr], default: None) –

set_data(*, *chat=None*, *user=None*, *data=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

update_data(*, *chat=None*, *user=None*, *data=None*, ***kwargs*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

reset_state(*, *chat=None*, *user=None*, *with_data=True*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- with_data (Optional[bool], default: True) –

classmethod check_address(*, *chat=None*, *user=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

Тип результата
(Union[str, int], Union[str, int])

finish(*, *chat=None*, *user=None*, *with_data=False*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- with_data (bool, default: False) –

reset_data(*, *chat=None*, *user=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

class MemoryStorage

close()

resolve_address(*chat*, *user*)

get_state(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Optional[str]

get_data(*, *chat=None*, *user=None*, *default=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- default (Optional[str], default: None) –

Тип результата
Dict

set_state(*, *chat=None*, *user=None*, *state=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- state (Optional[AnyStr], default: None) –

set_data(*, *chat=None*, *user=None*, *data=None*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –
- data (Optional[Dict], default: None) –

update_data(*, *chat=None*, *user=None*, *data=None*, ***kwargs*)

Параметры

- chat (Union[str, int, None], default: None) –
- user (Union[str, int, None], default: None) –

- **data** (Optional[Dict], default: None) –

reset_state(*, *chat=None*, *user=None*, *with_data=True*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **with_data** (Optional[bool], default: True) –

classmethod check_address(*, *chat=None*, *user=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –

Тип результата

(Union[str, int], Union[str, int])

finish(*, *chat=None*, *user=None*, *with_data=False*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –
- **with_data** (bool, default: False) –

reset_data(*, *chat=None*, *user=None*)

Параметры

- **chat** (Union[str, int, None], default: None) –
- **user** (Union[str, int, None], default: None) –

vk_maria.exceptions

Exceptions

AccessIsDeniedError	
AuthorizationError	
DeprecatedMethodError	
InvalidParametersError	
KeyIsNotValidError	
PermissionError	
ServerError	
UnknownError	
UnknownMethodError	
VkMariaException(code, text)	**param code**
WrongRequestError	

vk_maria.exceptions.AccessIsDeniedError

exception AccessIsDeniedError

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

vk_maria.exceptions.AuthorizationError

exception AuthorizationError

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

vk_maria.exceptions.DeprecatedMethodError

exception DeprecatedMethodError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

vk_maria.exceptions.InvalidParametersError

exception InvalidParametersError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

vk_maria.exceptions.KeyIsNotValidError

exception KeyIsNotValidError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

vk_maria.exceptions.PermissionError

exception PermissionError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

vk_maria.exceptions.ServerError

exception ServerError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

vk_maria.exceptions.UnknownError

exception UnknownError

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

vk_maria.exceptions.UnknownMethodError

exception UnknownMethodError

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

vk_maria.exceptions.VkMariaException

exception VkMariaException(*code*, *text*)

Параметры

- code (int) –
- text (str) –

exceptions = {1: <class 'vk_maria.exceptions.UnknownError'>, 3: <class 'vk_maria.exceptions.UnknownMethodError'>, 5: <class 'vk_maria.exceptions.AuthorizationError'>, 7: <class 'vk_maria.exceptions.PermissionError'>, 8: <class 'vk_maria.exceptions.WrongRequestError'>, 10: <class 'vk_maria.exceptions.ServerError'>, 15: <class 'vk_maria.exceptions.AccessIsDeniedError'>, 23: <class 'vk_maria.exceptions.DeprecatedMethodError'>, 27: <class 'vk_maria.exceptions.KeyIsNotValidError'>, 100: <class 'vk_maria.exceptions.InvalidParametersError'>}

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

vk_maria.exceptions.WrongRequestError

exception WrongRequestError

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception UnknownError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception UnknownMethodError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception AuthorizationError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception PermissionError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception WrongRequestError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception ServerError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception AccessIsDeniedError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception DeprecatedMethodError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception InvalidParametersError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception KeyIsNotValidError

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception VkMariaException(*code*, *text*)

Параметры

- code (int) –
- text (str) –

exceptions = {1: <class 'vk_maria.exceptions.UnknownError'>, 3: <class 'vk_maria.exceptions.UnknownMethodError'>, 5: <class 'vk_maria.exceptions.AuthorizationError'>, 7: <class 'vk_maria.exceptions.PermissionError'>, 8: <class 'vk_maria.exceptions.WrongRequestError'>, 10: <class 'vk_maria.exceptions.ServerError'>, 15: <class 'vk_maria.exceptions.AccessIsDeniedError'>, 23: <class 'vk_maria.exceptions.DeprecatedMethodError'>, 27: <class 'vk_maria.exceptions.KeyIsNotValidError'>, 100: <class 'vk_maria.exceptions.InvalidParametersError'>}

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

vk_maria.longpoll

vk_maria.longpoll.longpoll

vk_maria.longpoll.longpoll

Classes

LongPoll

vk_maria.longpoll.longpoll.LongPoll

class LongPoll(*vk*)

Базовые классы: object

Methods

listen

listen()

class LongPoll(*vk*)

listen()

vk_maria.mixins

Classes

Singleton

vk_maria.mixins.Singleton

class Singleton(**args*, ***kwargs*)

Базовые классы: object

Methods

class Singleton(**args*, ***kwargs*)

vk_maria.responses

Classes

DocsGetMessagesUploadServer
DocsGetWallUploadServer
DocsSave

continues on next page

Таблица 3 – продолжение с предыдущей страницы

DocsSearch
GroupsGetBanned
GroupsGetById
GroupsGetMembers
GroupsGetOnlineStatus
GroupsGetTokenPermissions
GroupsIsMember
MarketGetGroupOrders
MarketGetOrderItems
MessagesDeleteChatPhoto
MessagesDeleteConversation
MessagesGetByConversationMessageId
MessagesGetById
MessagesGetConversationMembers
MessagesGetConversations
MessagesGetConversationsById
MessagesGetHistory
MessagesGetHistoryAttachments
MessagesGetImportantMessages
MessagesGetIntentUsers
MessagesGetInviteLink
MessagesGetLongpollHistory
MessagesGetLongpollServer
MessagesIsMessagesFromGroupAllowed
MessagesPin
MessagesSearch

continues on next page

Таблица 3 – продолжение с предыдущей страницы

MessagesSearchConversations
MessagesSetChatPhoto
PhotosGetChatUploadServer
PhotosGetMessagesUploadServer
PhotosGetOwnerCoverPhotoUploadServer
PhotosSaveMessagesPhoto
PhotosSaveOwnerCoverPhoto
PodcastsSearchPodcast
Response
ResponseItem
StoriesGet
StoriesGetById
StoriesGetPhotoUploadServer
StoriesGetReplies
StoriesGetStats
StoriesGetVideoUploadServer
StoriesGetViewers
StoriesSave
UtilsCheckLink
UtilsGetLinkStats
UtilsGetShortLink
UtilsResolveScreenName
WallCreateComment

vk_maria.responses.DocsGetMessagesUploadServer

class DocsGetMessagesUploadServer(*response*)

Базовые классы: *Response*

Methods

Attributes

upload_url

upload_url: str

vk_maria.responses.DocsGetWallUploadServer

class DocsGetWallUploadServer(*response*)

Базовые классы: *Response*

Methods

Attributes

upload_url

upload_url: str

vk_maria.responses.DocsSave

class DocsSave(*response*)

Базовые классы: *Response*, *Document*

Methods

Attributes

id: int

owner_id: int

title: str

size: int

ext: str

url: str

date: int

type: int

preview: Dict

vk_maria.responses.DocsSearch

class DocsSearch(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items

count: int

items: List[Dict]

vk_maria.responses.GroupsGetBanned

class GroupsGetBanned(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items

count: int

items: List[Dict]

vk_maria.responses.GroupsGetById

class GroupsGetById(**args*)

Базовые классы: *Response*, *Group*

Methods

Attributes

activity
addresses
age_limits
ban_info
can_create_topic
can_message
can_post

continues on next page

Таблица 4 – продолжение с предыдущей страницы

can_see_all_posts
can_upload_doc
can_upload_video
city
contacts
counters
country
cover
crop_photo
description
finish_date
fixed_post
has_photo
is_favorite
is_hidden_from_feed
is_messages_blocked
links
main_album_id
main_section
market
member_status
members_count
place
public_date_label
site
start_date

continues on next page

Таблица 4 – продолжение с предыдущей страницы

status
trending
verified
wall
wiki_page

```
activity: str = None
addresses: Adresses = None
age_limits: int = None
ban_info: BanInfo = None
can_create_topic: int = None
can_message: int = None
can_post: int = None
can_see_all_posts: int = None
can_upload_doc: int = None
can_upload_video: int = None
city: City = None
contacts: Contacts = None
counters: Counters = None
country: Country = None
cover: Cover = None
crop_photo: Dict = None
description: str = None
finish_date: int = None
fixed_post: int = None
has_photo: int = None
is_favorite: int = None
is_hidden_from_feed: int = None
is_messages_blocked: int = None
```

```
links: Links = None
main_album_id: int = None
main_section: int = None
market: Market = None
member_status: int = None
members_count: int = None
place: Place = None
public_date_label: str = None
site: str = None
start_date: int = None
status: str = None
trending: int = None
verified: int = None
wall: int = None
wiki_page: str = None
id: int
name: str
screen_name: str
is_closed: int
deactivated: str
is_admin: int
admin_level: int
is_member: int
is_advertiser: int
invited_by: int
type: str
photo_50: str
photo_100: str
photo_200: str
```

vk_maria.responses.GroupsGetMembers

class GroupsGetMembers(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items

count: int

items: List[int]

vk_maria.responses.GroupsGetOnlineStatus

class GroupsGetOnlineStatus(*response*)

Базовые классы: *Response*

Methods

Attributes

status
minutes

status: str

minutes: int

vk_maria.responses.GroupsGetTokenPermissions

class GroupsGetTokenPermissions(*response*)

Базовые классы: *Response*

Methods

Attributes

mask
settings

mask: int

settings: List[Dict]

vk_maria.responses.GroupsIsMember

class GroupsIsMember(*response*)

Базовые классы: *Response*

Methods

Attributes

member
request
invitation
can_invite
can_recall
user_id

member: int

request: int

invitation: int

can_invite: int

can_recall: int

user_id: int

vk_maria.responses.MarketGetGroupOrders

class MarketGetGroupOrders(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items

count: int

items: List[Dict]

vk_maria.responses.MarketGetOrderItems

class MarketGetOrderItems(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items

count: int

items: List[Dict]

vk_maria.responses.MessagesDeleteChatPhoto

class MessagesDeleteChatPhoto(*response*)

Базовые классы: *Response*

Methods

Attributes

message_id
chat

message_id: int

chat: dict

vk_maria.responses.MessagesDeleteConversation

class MessagesDeleteConversation(*response*)

Базовые классы: *Response*

Methods

Attributes

last_deleted_id

last_deleted_id: int

vk_maria.responses.MessagesGetByConversationMessageId

class MessagesGetByConversationMessageId(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items

count: int

items: List[Dict]

vk_maria.responses.MessagesGetById

class MessagesGetById(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items

count: int

items: List[Dict]

vk_maria.responses.MessagesGetConversationMembers

class MessagesGetConversationMembers(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items
profiles
groups

count: int

items: List[Dict]

profiles: List[*Profile*]

groups: List[*Group*]

vk_maria.responses.MessagesGetConversations

class MessagesGetConversations(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items
unread_count
profiles
groups

count: int

items: List[Dict]

unread_count: int

profiles: List[*Profile*]

groups: List[*Group*]

vk_maria.responses.MessagesGetConversationsById

class MessagesGetConversationsById(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items

count: int

items: List[Dict]

vk_maria.responses.MessagesGetHistory

class MessagesGetHistory(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items
in_read
out_read

count: int

items: Dict

in_read: int

out_read: int

vk_maria.responses.MessagesGetHistoryAttachments

class MessagesGetHistoryAttachments(*response*)

Базовые классы: *Response*

Methods

Attributes

items
next_from

items: List[Dict]

next_from: str

vk_maria.responses.MessagesGetImportantMessages

class MessagesGetImportantMessages(*response*)

Базовые классы: *Response*

Methods

Attributes

messages

messages: Dict

vk_maria.responses.MessagesGetIntentUsers

class MessagesGetIntentUsers(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items

count: int

items: List[int]

vk_maria.responses.MessagesGetInviteLink

class MessagesGetInviteLink(*response*)

Базовые классы: *Response*

Methods

Attributes

link

link: str

vk_maria.responses.MessagesGetLongpollHistory

class MessagesGetLongpollHistory(*response*)

Базовые классы: *Response*

Methods

Attributes

history
messages
groups
profiles

history: List

messages: List

groups: List[*Group*]

profiles: List[*Profile*]

vk_maria.responses.MessagesGetLongpollServer

class MessagesGetLongpollServer(*response*)

Базовые классы: *Response*

Methods

Attributes

key
server
ts

key: str

server: str

ts: int

vk_maria.responses.MessagesIsMessagesFromGroupAllowed

class MessagesIsMessagesFromGroupAllowed(*response*)

Базовые классы: *Response*

Methods

Attributes

is_allowed

is_allowed: int

vk_maria.responses.MessagesPin

class MessagesPin(*response*)

Базовые классы: *Response*

Methods

Attributes

id
date
from_id
text
attachments
geo
fwd_messages

id: int

date: int

from_id: int

text: str

attachments: List[Dict]

geo: Dict

fwd_messages: List[Dict]

vk_maria.responses.MessagesSearch

class MessagesSearch(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items

count: int

items: List[Dict]

vk_maria.responses.MessagesSearchConversations

class MessagesSearchConversations(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items

count: int

items: List[Dict]

vk_maria.responses.MessagesSetChatPhoto

class MessagesSetChatPhoto(*response*)

Базовые классы: *Response*

Methods

Attributes

message_id
chat

message_id: int

chat: Dict

vk_maria.responses.PhotosGetChatUploadServer

class PhotosGetChatUploadServer(*response*)

Базовые классы: *Response*

Methods

Attributes

upload_url

upload_url: str

vk_maria.responses.PhotosGetMessagesUploadServer

class PhotosGetMessagesUploadServer(*response*)

Базовые классы: *Response*

Methods

Attributes

upload_url
album_id
group_id

upload_url: str

album_id: int

group_id: int

vk_maria.responses.PhotosGetOwnerCoverPhotoUploadServer

class PhotosGetOwnerCoverPhotoUploadServer(*response*)

Базовые классы: *Response*

Methods

Attributes

upload_url

upload_url: str

vk_maria.responses.PhotosSaveMessagesPhoto

class PhotosSaveMessagesPhoto(*response*)

Базовые классы: *Response*

Methods

Attributes

id
pid
aid
owner_id
src
src_big
src_small
created
src_xbig
src_xxbig

id: int

pid: int

aid: int

owner_id: int

src: str

src_big: str

src_small: str

created: str

src_xbig: str

src_xxbig: str

vk_maria.responses.PhotosSaveOwnerCoverPhoto

class PhotosSaveOwnerCoverPhoto(*response*)

Базовые классы: *Response*

Methods

Attributes

images

images: List[Dict]

vk_maria.responses.PodcastsSearchPodcast

class PodcastsSearchPodcast(*response*)

Базовые классы: *Response*

Methods

Attributes

results_total
podcasts

results_total: int

podcasts: List[Dict]

vk_maria.responses.Response

class Response(*response*)

Базовые классы: object

Methods

vk_maria.responses.ResponseItem

class ResponseItem(***kwargs*)

Базовые классы: object

Methods

vk_maria.responses.StoriesGet

class StoriesGet(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items
profiles
groups

count: int

items: List[Dict]

profiles: List[Dict]

groups: List[Dict]

vk_maria.responses.StoriesGetById

class StoriesGetById(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items
profiles
groups

count: int

items: List[Dict]

profiles: List[Dict]

groups: List[Dict]

vk_maria.responses.StoriesGetPhotoUploadServer

class StoriesGetPhotoUploadServer(*response*)

Базовые классы: *Response*

Methods

Attributes

upload_result

upload_result: str

vk_maria.responses.StoriesGetReplies

class StoriesGetReplies(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items
profiles
groups

count: int

items: List[Dict]

profiles: List[Dict]

groups: List[Dict]

vk_maria.responses.StoriesGetStats

class StoriesGetStats(*response*)

Базовые классы: *Response*

Methods

Attributes

views
replies
answer
shares
subscribers
bans
open_link

views: Dict

replies: Dict

answer: Dict

shares: Dict

subscribers: Dict

bans: Dict

open_link: Dict

vk_maria.responses.StoriesGetVideoUploadServer

class StoriesGetVideoUploadServer(*response*)

Базовые классы: *Response*

Methods

Attributes

upload_result

upload_result: str

vk_maria.responses.StoriesGetViewers

class StoriesGetViewers(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items

count: int

items: List[Dict]

vk_maria.responses.StoriesSave

class StoriesSave(*response*)

Базовые классы: *Response*

Methods

Attributes

count
items

count: int

items: List[Dict]

vk_maria.responses.UtilsCheckLink

class UtilsCheckLink(*response*)

Базовые классы: *Response*

Methods

Attributes

status
link

status: str

link: str

vk_maria.responses.UtilsGetLinkStats

class UtilsGetLinkStats(*response*)

Базовые классы: *Response*

Methods

Attributes

key
stats

key: str

stats: List[Dict]

vk_maria.responses.UtilsGetShortLink

class UtilsGetShortLink(*response*)

Базовые классы: *Response*

Methods

Attributes

short_url
access_key
key
url

short_url: str

access_key: str

key: str

url: str

vk_maria.responses.UtilsResolveScreenName

class UtilsResolveScreenName(*response*)

Базовые классы: *Response*

Methods

Attributes

type
object_id

type: str

object_id: int

vk_maria.responses.WallCreateComment

class WallCreateComment(*response*)

Базовые классы: *Response*

Methods

Attributes

comment_id
parent_stack

comment_id: int

parent_stack: List[int]

class ResponseItem(***kwargs*)

class Response(*response*)

class GroupsGetBanned(*response*)

count: int

items: List[Dict]

class GroupsGetById(**args*)

activity: str = None

addresses: *Adresses* = None

age_limits: int = None

ban_info: *BanInfo* = None

can_create_topic: int = None

can_message: int = None

can_post: int = None

can_see_all_posts: int = None

can_upload_doc: int = None

can_upload_video: int = None

city: *City* = None

contacts: *Contacts* = None

counters: *Counters* = None

country: *Country* = None

cover: *Cover* = None

crop_photo: Dict = None

description: str = None

finish_date: int = None

fixed_post: int = None

has_photo: int = None

is_favorite: int = None

is_hidden_from_feed: int = None

is_messages_blocked: int = None

links: *Links* = None

main_album_id: int = None

main_section: int = None

market: *Market* = None

member_status: int = None

members_count: int = None

place: *Place* = None

public_date_label: str = None

site: str = None

start_date: int = None

status: str = None

trending: int = None

verified: int = None

wall: int = None

wiki_page: str = None

id: int

name: str

screen_name: str

is_closed: int

deactivated: str

is_admin: int

admin_level: int

is_member: int

is_advertiser: int

invited_by: int

type: str

photo_50: str

photo_100: str

photo_200: str

class GroupsGetMembers(*response*)

count: int

items: List[int]

class GroupsIsMember(*response*)

member: int

request: int

invitation: int

can_invite: int

can_recall: int

user_id: int

class GroupsGetOnlineStatus(*response*)

status: str

minutes: int

class GroupsGetTokenPermissions(*response*)

mask: int

settings: List[Dict]

class DocsGetMessagesUploadServer(*response*)

upload_url: str

class DocsGetWallUploadServer(*response*)

upload_url: str

class DocsSearch(*response*)

count: int

items: List[Dict]

class DocsSave(*response*)

id: int

owner_id: int

title: str

size: int

ext: str

url: str

date: int

type: int

preview: Dict

class MessagesDeleteChatPhoto(*response*)

message_id: int

chat: dict

class MessagesDeleteConversation(*response*)

last_deleted_id: int

class MessagesGetConversationMembers(*response*)

count: int

items: List[Dict]

profiles: List[*Profile*]

groups: List[*Group*]

class MessagesGetByConversationMessageId(*response*)

count: int

items: List[Dict]

class MessagesGetById(*response*)

count: int

items: List[Dict]

class MessagesGetConversations(*response*)

count: int

items: List[Dict]

```
    unread_count: int
    profiles: List[Profile]
    groups: List[Group]
class MessagesGetConversationsById(response)
    count: int
    items: List[Dict]
class MessagesGetHistory(response)
    count: int
    items: Dict
    in_read: int
    out_read: int
class MessagesGetHistoryAttachments(response)
    items: List[Dict]
    next_from: str
class MessagesGetImportantMessages(response)
    messages: Dict
class MessagesGetIntentUsers(response)
    count: int
    items: List[int]
class MessagesGetInviteLink(response)
    link: str
class MessagesIsMessagesFromGroupAllowed(response)
    is_allowed: int
class MessagesPin(response)
    id: int
    date: int
    from_id: int
    text: str
    attachments: List[Dict]
    geo: Dict
    fwd_messages: List[Dict]
```

class MessagesSearch(*response*)

count: int

items: List[Dict]

class MessagesSearchConversations(*response*)

count: int

items: List[Dict]

class MessagesSetChatPhoto(*response*)

message_id: int

chat: Dict

class MessagesGetLongpollServer(*response*)

key: str

server: str

ts: int

class MessagesGetLongpollHistory(*response*)

history: List

messages: List

groups: List[*Group*]

profiles: List[*Profile*]

class MarketGetGroupOrders(*response*)

count: int

items: List[Dict]

class MarketGetOrderItems(*response*)

count: int

items: List[Dict]

class PhotosGetChatUploadServer(*response*)

upload_url: str

class PhotosGetMessagesUploadServer(*response*)

upload_url: str

album_id: int

group_id: int

class PhotosGetOwnerCoverPhotoUploadServer(*response*)

upload_url: str

class PhotosSaveMessagesPhoto(*response*)

id: int

pid: int

aid: int

owner_id: int

src: str

src_big: str

src_small: str

created: str

src_xbig: str

src_xxbig: str

class PhotosSaveOwnerCoverPhoto(*response*)

images: List[Dict]

class PodcastsSearchPodcast(*response*)

results_total: int

podcasts: List[Dict]

class StoriesGet(*response*)

count: int

items: List[Dict]

profiles: List[Dict]

groups: List[Dict]

class StoriesGetById(*response*)

count: int

items: List[Dict]

profiles: List[Dict]

groups: List[Dict]

class StoriesGetPhotoUploadServer(*response*)

upload_result: str

class StoriesGetReplies(*response*)

count: int

items: List[Dict]

profiles: List[Dict]

groups: List[Dict]

class StoriesGetStats(*response*)

views: Dict

replies: Dict

answer: Dict

shares: Dict

subscribers: Dict

bans: Dict

open_link: Dict

class StoriesGetVideoUploadServer(*response*)

upload_result: str

class StoriesGetViewers(*response*)

count: int

items: List[Dict]

class StoriesSave(*response*)

count: int

items: List[Dict]

class UtilsCheckLink(*response*)

status: str

link: str

class UtilsGetLinkStats(*response*)

key: str

stats: List[Dict]

class UtilsGetShortLink(*response*)

short_url: str

access_key: str

key: str

url: str

class UtilsResolveScreenName(*response*)

type: str

object_id: int

class WallCreateComment(*response*)

comment_id: int

parent_stack: List[int]

vk_maria.types

vk_maria.types.callback_query_event
vk_maria.types.chat
vk_maria.types.event
vk_maria.types.event_type
vk_maria.types.keyboard
vk_maria.types.message
vk_maria.types.message_event
vk_maria.types.state

vk_maria.types.callback_query_event

Classes

CallbackQueryEvent

vk_maria.types.callback_query_event.CallbackQueryEvent

class CallbackQueryEvent(*vk*, *raw*)

Базовые классы: *Event*, *CallbackQuery*

Methods

answer	param event_data

Attributes

answer(*event_data=None*)

Параметры
event_data (Optional[dict], default: None) –

user_id: int

peer_id: int

event_id: str

payload: DotDict

conversation_message_id: int

type: str

class CallbackQueryEvent(*vk*, *raw*)

answer(*event_data=None*)

Параметры
event_data (Optional[dict], default: None) –

user_id: int

peer_id: int

event_id: str

payload: DotDict

conversation_message_id: int

type: str

vk_maria.types.chat

Classes

Chat	

vk_maria.types.chat.Chat

class Chat

Базовые классы: object

Methods

get_chat_id	
get_user_id	
resolve_address	**param event**
set	**param chat_id**

Attributes

chat_id
user_id

chat_id: int = None

user_id: int = None

classmethod get_chat_id()

classmethod get_user_id()

classmethod set(*chat_id*, *user_id*)

Параметры

- chat_id (int) –
- user_id (int) –

static resolve_address(*event*)

Параметры

event (*Event*) –

Тип результата

(Optional[int], Optional[int])

class Chat

chat_id: int = None

user_id: int = None

classmethod get_chat_id()

classmethod get_user_id()

classmethod set(*chat_id*, *user_id*)

Параметры

- chat_id (int) –
- user_id (int) –

static resolve_address(*event*)

Параметры
event (*Event*) –

Тип результата
(Optional[int], Optional[int])

vk_maria.types.event

Classes

Event	

vk_maria.types.event.Event

class Event(*vk*, *raw*)

Базовые классы: object

Methods

class Event(*vk*, *raw*)

vk_maria.types.event_type

Classes

EventType	An enumeration.

vk_maria.types.event_type.EventType

class EventType(*value*)

Базовые классы: Enum

An enumeration.

Attributes

MESSAGE_NEW
MESSAGE_REPLY
MESSAGE_EDIT
MESSAGE_EVENT
MESSAGE_TYPING_STATE
MESSAGE_ALLOW
MESSAGE_DENY
PHOTO_NEW
PHOTO_COMMENT_NEW
PHOTO_COMMENT_EDIT
PHOTO_COMMENT_RESTORE
PHOTO_COMMENT_DELETE
AUDIO_NEW
VIDEO_NEW
VIDEO_COMMENT_NEW
VIDEO_COMMENT_EDIT
VIDEO_COMMENT_RESTORE
VIDEO_COMMENT_DELETE
WALL_POST_NEW
WALL_REPOST
WALL_REPLY_NEW

continues on next page

Таблица 5 – продолжение с предыдущей страницы

WALL_REPLY_EDIT
WALL_REPLY_RESTORE
WALL_REPLY_DELETE
BOARD_POST_NEW
BOARD_POST_EDIT
BOARD_POST_RESTORE
BOARD_POST_DELETE
MARKET_COMMENT_NEW
MARKET_COMMENT_EDIT
MARKET_COMMENT_RESTORE
MARKET_COMMENT_DELETE
GROUP_LEAVE
GROUP_JOIN
USER_BLOCK
USER_UNBLOCK
POLL_VOTE_NEW
GROUP_OFFICERS_EDIT
GROUP_CHANGE_SETTINGS
GROUP_CHANGE_PHOTO
VKPAY_TRANSACTION

```
MESSAGE_NEW = 'message_new'

MESSAGE_REPLY = 'message_reply'

MESSAGE_EDIT = 'message_edit'

MESSAGE_EVENT = 'message_event'

MESSAGE_TYPING_STATE = 'message_typing_state'

MESSAGE_ALLOW = 'message_allow'
```

```
MESSAGE_DENY = 'message_deny'

PHOTO_NEW = 'photo_new'

PHOTO_COMMENT_NEW = 'photo_comment_new'

PHOTO_COMMENT_EDIT = 'photo_comment_edit'

PHOTO_COMMENT_RESTORE = 'photo_comment_restore'

PHOTO_COMMENT_DELETE = 'photo_comment_delete'

AUDIO_NEW = 'audio_new'

VIDEO_NEW = 'video_new'

VIDEO_COMMENT_NEW = 'video_comment_new'

VIDEO_COMMENT_EDIT = 'video_comment_edit'

VIDEO_COMMENT_RESTORE = 'video_comment_restore'

VIDEO_COMMENT_DELETE = 'video_comment_delete'

WALL_POST_NEW = 'wall_post_new'

WALL_REPOST = 'wall_repost'

WALL_REPLY_NEW = 'wall_reply_new'

WALL_REPLY_EDIT = 'wall_reply_edit'

WALL_REPLY_RESTORE = 'wall_reply_restore'

WALL_REPLY_DELETE = 'wall_reply_delete'

BOARD_POST_NEW = 'board_post_new'

BOARD_POST_EDIT = 'board_post_edit'

BOARD_POST_RESTORE = 'board_post_restore'

BOARD_POST_DELETE = 'board_post_delete'

MARKET_COMMENT_NEW = 'market_comment_new'

MARKET_COMMENT_EDIT = 'market_comment_edit'

MARKET_COMMENT_RESTORE = 'market_comment_restore'

MARKET_COMMENT_DELETE = 'market_comment_delete'

GROUP_LEAVE = 'group_leave'

GROUP_JOIN = 'group_join'

USER_BLOCK = 'user_block'

USER_UNBLOCK = 'user_unblock'

POLL_VOTE_NEW = 'poll_vote_new'
```

GROUP_OFFICERS_EDIT = 'group_officers_edit'

GROUP_CHANGE_SETTINGS = 'group_change_settings'

GROUP_CHANGE_PHOTO = 'group_change_photo'

VKPAY_TRANSACTION = 'vkpay_transaction'

class EventType(*value*)

An enumeration.

MESSAGE_NEW = 'message_new'

MESSAGE_REPLY = 'message_reply'

MESSAGE_EDIT = 'message_edit'

MESSAGE_EVENT = 'message_event'

MESSAGE_TYPING_STATE = 'message_typing_state'

MESSAGE_ALLOW = 'message_allow'

MESSAGE_DENY = 'message_deny'

PHOTO_NEW = 'photo_new'

PHOTO_COMMENT_NEW = 'photo_comment_new'

PHOTO_COMMENT_EDIT = 'photo_comment_edit'

PHOTO_COMMENT_RESTORE = 'photo_comment_restore'

PHOTO_COMMENT_DELETE = 'photo_comment_delete'

AUDIO_NEW = 'audio_new'

VIDEO_NEW = 'video_new'

VIDEO_COMMENT_NEW = 'video_comment_new'

VIDEO_COMMENT_EDIT = 'video_comment_edit'

VIDEO_COMMENT_RESTORE = 'video_comment_restore'

VIDEO_COMMENT_DELETE = 'video_comment_delete'

WALL_POST_NEW = 'wall_post_new'

WALL_REPOST = 'wall_repost'

WALL_REPLY_NEW = 'wall_reply_new'

WALL_REPLY_EDIT = 'wall_reply_edit'

WALL_REPLY_RESTORE = 'wall_reply_restore'

WALL_REPLY_DELETE = 'wall_reply_delete'

BOARD_POST_NEW = 'board_post_new'

```
    BOARD_POST_EDIT = 'board_post_edit'

    BOARD_POST_RESTORE = 'board_post_restore'

    BOARD_POST_DELETE = 'board_post_delete'

    MARKET_COMMENT_NEW = 'market_comment_new'

    MARKET_COMMENT_EDIT = 'market_comment_edit'

    MARKET_COMMENT_RESTORE = 'market_comment_restore'

    MARKET_COMMENT_DELETE = 'market_comment_delete'

    GROUP_LEAVE = 'group_leave'

    GROUP_JOIN = 'group_join'

    USER_BLOCK = 'user_block'

    USER_UNBLOCK = 'user_unblock'

    POLL_VOTE_NEW = 'poll_vote_new'

    GROUP_OFFICERS_EDIT = 'group_officers_edit'

    GROUP_CHANGE_SETTINGS = 'group_change_settings'

    GROUP_CHANGE_PHOTO = 'group_change_photo'

    VKPAY_TRANSACTION = 'vkpay_transaction'
```

vk_maria.types.keyboard

Functions

construct_json	
unpack_button	**param button**

vk_maria.types.keyboard.construct_json

construct_json(*model_dict*)

vk_maria.types.keyboard.unpack_button

unpack_button(*button*)

Параметры
button (*BaseButton*) –

Classes

BaseButton	
Button	
CallbackButton	**param color**
Color	An enumeration.
KeyboardMarkup	**param inline**
KeyboardModel	
KeyboardModelMeta	
LocationButton	**param payload**
OpenLinkButton	**param link**
TextButton	**param color**
VKAppsButton	**param app_id**
VKPayButton	**param payload**

vk_maria.types.keyboard.BaseButton

class BaseButton

Базовые классы: object

Methods

Attributes

type
action

type: str

action: dict

vk_maria.types.keyboard.Button

class Button

Базовые классы: object

Methods

Text
: alias of *TextButton*

OpenLink
: alias of *OpenLinkButton*

Location
: alias of *LocationButton*

VKPay
: alias of *VKPayButton*

VKApps
: alias of *VKAppsButton*

Callback
: alias of *CallbackButton*

vk_maria.types.keyboard.CallbackButton

class CallbackButton(*color*, *label*, *payload*)

Базовые классы: *BaseButton*

Параметры

- color (str) –
- label (str) –
- payload (dict) –

Methods

Attributes

type

type: str = 'callback'

action: dict

vk_maria.types.keyboard.Color

class Color(*value*)

Базовые классы: str, Enum

An enumeration.

Attributes

PRIMARY
SECONDARY
NEGATIVE
POSITIVE

PRIMARY = 'primary'

SECONDARY = 'secondary'

NEGATIVE = 'negative'

```
POSITIVE = 'positive'
```

vk_maria.types.keyboard.KeyboardMarkup

class KeyboardMarkup(*inline=False*, *one_time=False*, *keyboard=None*)

Базовые классы: object

Параметры

- **inline** (bool, default: False) –
- **one_time** (bool, default: False) –
- **keyboard** (Optional[List[List[*BaseButton*]]], default: None) –

Methods

add_button
add_row
get_json

add_button(*button*)

add_row()

get_json()

vk_maria.types.keyboard.KeyboardModel

class KeyboardModel

Базовые классы: object

Methods

Attributes

inline
one_time
row1
row2
row3
row4
row5

inline: bool = False

one_time: bool = False

row1: List[*Button*] = None

row2: List[*Button*] = None

row3: List[*Button*] = None

row4: List[*Button*] = None

row5: List[*Button*] = None

vk_maria.types.keyboard.KeyboardModelMeta

class KeyboardModelMeta(*name*, *bases*, *namespace*)

Базовые классы: type

Methods

mro	Return a type's method resolution order.

__call__(**args*, ***kwargs*)

Call self as a function.

mro()

Return a type's method resolution order.

vk_maria.types.keyboard.LocationButton

class LocationButton(*payload*)

Базовые классы: *BaseButton*

Параметры

payload (dict) –

Methods

Attributes

type

type: str = 'location'

action: dict

vk_maria.types.keyboard.OpenLinkButton

class OpenLinkButton(*link*, *label*, *payload=None*)

Базовые классы: *BaseButton*

Параметры

- link (str) –
- label (str) –
- payload (Optional[dict], default: None) –

Methods

Attributes

type

type: str = 'open_link'

action: dict

vk_maria.types.keyboard.TextButton

class TextButton(*color*, *label*, *payload=None*)

Базовые классы: *BaseButton*

Параметры

- color (str) –
- label (str) –
- payload (Optional[dict], default: None) –

Methods

Attributes

type

type: str = 'text'

action: dict

vk_maria.types.keyboard.VKAppsButton

class VKAppsButton(*app_id*, *owner_id*, *label*, *hash_*, *payload=None*)

Базовые классы: *BaseButton*

Параметры

- app_id (int) –
- owner_id (int) –
- label (str) –
- hash_ (str) –
- payload (Optional[dict], default: None) –

Methods

Attributes

type

type: str = 'open_app'

action: dict

vk_maria.types.keyboard.VKPayButton

class VKPayButton(*payload*, *hash_*)

Базовые классы: *BaseButton*

Параметры

- payload (dict) –
- hash_ (str) –

Methods

Attributes

type

type: str = 'vkpay'

action: dict

class BaseButton

type: str

action: dict

class TextButton(*color*, *label*, *payload=None*)

Параметры

- color (str) –
- label (str) –
- payload (Optional[dict], default: None) –

type: str = 'text'

action: dict

class OpenLinkButton(*link*, *label*, *payload=None*)

Параметры

- link (str) –
- label (str) –
- payload (Optional[dict], default: None) –

type: str = 'open_link'

action: dict

class LocationButton(*payload*)

Параметры

payload (dict) –

type: str = 'location'

action: dict

class VKPayButton(*payload*, *hash_*)

Параметры

- payload (dict) –
- hash_ (str) –

type: str = 'vkpay'

action: dict

class VKAppsButton(*app_id*, *owner_id*, *label*, *hash_*, *payload=None*)

Параметры

- app_id (int) –
- owner_id (int) –
- label (str) –
- hash_ (str) –
- payload (Optional[dict], default: None) –

type: str = 'open_app'

action: dict

class CallbackButton(*color*, *label*, *payload*)

Параметры

- color (str) –
- label (str) –
- payload (dict) –

type: str = 'callback'

action: dict

class Button

Text
alias of *TextButton*

OpenLink
alias of *OpenLinkButton*

Location
alias of *LocationButton*

VKPay
alias of *VKPayButton*

VKApps
alias of *VKAppsButton*

Callback
alias of *CallbackButton*

class Color(*value*)

An enumeration.

PRIMARY = 'primary'

SECONDARY = 'secondary'

NEGATIVE = 'negative'

POSITIVE = 'positive'

class KeyboardModelMeta(*name*, *bases*, *namespace*)

mro()
Return a type's method resolution order.

class KeyboardModel

inline: bool = False

one_time: bool = False

row1: List[*Button*] = None

row2: List[*Button*] = None

row3: List[*Button*] = None

row4: List[*Button*] = None

row5: List[*Button*] = None

unpack_button(*button*)

Параметры
button (*BaseButton*) –

construct_json(*model_ dict*)

class KeyboardMarkup(*inline=False*, *one_time=False*, *keyboard=None*)

Параметры

- inline (bool, default: False) –
- one_time (bool, default: False) –
- keyboard (Optional[List[List[*BaseButton*]]], default: None) –

add_button(*button*)

add_row()

get_json()

vk_maria.types.message

Classes

BaseEvent	
CallbackQuery	Описывает события типа MESSAGE_EVENT
Message	Описывает события типа MESSAGE_NEW
MessageInfo	

vk_maria.types.message.BaseEvent

class BaseEvent

Базовые классы: object

Methods

Attributes

type

type: str

vk_maria.types.message.CallbackQuery

class CallbackQuery

Базовые классы: *BaseEvent*

Описывает события типа MESSAGE_EVENT

Methods

answer	**param event_data**

Attributes

user_id
peer_id
event_id
payload
conversation_message_id

user_id: int

peer_id: int

event_id: str

payload: DotDict

conversation_message_id: int

answer(*event_data=None*)

Параметры

event_data (Optional[dict], default: None) –

type: str

vk_maria.types.message.Message

class Message

Базовые классы: *BaseEvent*

Описывает события типа MESSAGE_NEW

Methods

answer	**param message**
reply	Не работает в беседах из-за ограниченного апи

Attributes

message
from_user
from_chat
from_group
chat_id
peer_id

message: *MessageInfo*

from_user: bool

from_chat: bool

from_group: bool

chat_id: Optional[int]

peer_id: int

answer(*message=None*, *domain=None*, *lat=None*, *long=None*, *attachment=None*, *reply_to=None*, *forward_messages=None*, *forward=None*, *sticker_id=None*, *keyboard=None*, *template=None*, *payload=None*, *content_source=None*, *dont_parse_links=None*, *disable_mentions=None*, *intent='default'*, *subscribe_id=None*)

Параметры

- message (Optional[str], default: None) –
- domain (Optional[str], default: None) –
- lat (Optional[float], default: None) –

- long (Optional[float], default: None) –
- reply_to (Optional[int], default: None) –
- forward_messages (Optional[List[int]], default: None) –
- sticker_id (Optional[int], default: None) –
- template (Optional[Dict], default: None) –
- content_source (Optional[Dict], default: None) –
- dont_parse_links (Optional[int], default: None) –
- disable_mentions (Optional[int], default: None) –
- intent (str, default: 'default') –
- subscribe_id (Optional[int], default: None) –

reply(*message=None*, *domain=None*, *lat=None*, *long=None*, *attachment=None*, *reply_to=None*, *forward_messages=None*, *forward=None*, *sticker_id=None*, *keyboard=None*, *template=None*, *payload=None*, *content_source=None*, *dont_parse_links=None*, *disable_mentions=None*, *intent='default'*, *subscribe_id=None*)

Не работает в беседах из-за ограниченного апи

Параметры

- message (Optional[str], default: None) –
- domain (Optional[str], default: None) –
- lat (Optional[float], default: None) –
- long (Optional[float], default: None) –
- reply_to (Optional[int], default: None) –
- forward_messages (Optional[List[int]], default: None) –
- sticker_id (Optional[int], default: None) –
- template (Optional[Dict], default: None) –
- content_source (Optional[Dict], default: None) –
- dont_parse_links (Optional[int], default: None) –
- disable_mentions (Optional[int], default: None) –
- intent (str, default: 'default') –
- subscribe_id (Optional[int], default: None) –

type: str

vk_maria.types.message.MessageInfo

class MessageInfo(**args*, ***kwargs*)

Базовые классы: DotDict

Methods

clear	
copy	
empty	
fromkeys	Create a new ordered dictionary with keys from iterable and values set to value.
get	
has_key	
items	
keys	
move_to_end	Move an existing element to the end (or beginning if last is false).
pop	If key is not found, d is returned if given, otherwise KeyError is raised.
popitem	as a 2-tuple; but raise KeyError if D is empty.
setdefault	
to_dict	
update	If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v
values	

Attributes

date
from_id
id
out
peer_id
text
conversation_message_id
fwd_messages
important
random_id
attachments
is_hidden

date: int

from_id: int

id: int

out: int

peer_id: int

text: str

conversation_message_id: int

fwd_messages: list

important: bool

random_id: int

attachments: list

is_hidden: bool

clear() → None. Remove all items from D.

copy() → a shallow copy of od

empty()

classmethod fromkeys(*seq*, *value=None*)

Create a new ordered dictionary with keys from iterable and values set to value.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

has_key(*key*)

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

move_to_end(*key*, *last=True*)

Move an existing element to the end (or beginning if last is false).

Raise KeyError if the element does not exist.

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair

as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict()

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.

If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

class BaseEvent

type: str

class MessageInfo(**args*, ***kwargs*)

date: int

from_id: int

id: int

out: int

peer_id: int

text: str

conversation_message_id: int

fwd_messages: list

important: bool

random_id: int

attachments: list

is_hidden: bool

clear() → None. Remove all items from D.

copy() → a shallow copy of od

empty()

classmethod fromkeys(*seq*, *value=None*)
Create a new ordered dictionary with keys from iterable and values set to value.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

has_key(*key*)

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

move_to_end(*key*, *last=True*)
Move an existing element to the end (or beginning if last is false).
Raise KeyError if the element does not exist.

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict()

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

values() → an object providing a view on D's values

class Message
Описывает события типа MESSAGE_NEW

message: *MessageInfo*

from_user: bool

from_chat: bool

from_group: bool

chat_id: Optional[int]

peer_id: int

answer(*message=None*, *domain=None*, *lat=None*, *long=None*, *attachment=None*, *reply_to=None*, *forward_messages=None*, *forward=None*, *sticker_id=None*, *keyboard=None*, *template=None*, *payload=None*, *content_source=None*, *dont_parse_links=None*, *disable_mentions=None*, *intent='default'*, *subscribe_id=None*)

Параметры

- message (Optional[str], default: None) –
- domain (Optional[str], default: None) –
- lat (Optional[float], default: None) –
- long (Optional[float], default: None) –
- reply_to (Optional[int], default: None) –
- forward_messages (Optional[List[int]], default: None) –
- sticker_id (Optional[int], default: None) –
- template (Optional[Dict], default: None) –
- content_source (Optional[Dict], default: None) –
- dont_parse_links (Optional[int], default: None) –
- disable_mentions (Optional[int], default: None) –
- intent (str, default: 'default') –
- subscribe_id (Optional[int], default: None) –

reply(*message=None*, *domain=None*, *lat=None*, *long=None*, *attachment=None*, *reply_to=None*, *forward_messages=None*, *forward=None*, *sticker_id=None*, *keyboard=None*, *template=None*, *payload=None*, *content_source=None*, *dont_parse_links=None*, *disable_mentions=None*, *intent='default'*, *subscribe_id=None*)

Не работает в беседах из-за ограниченного апи

Параметры

- message (Optional[str], default: None) –
- domain (Optional[str], default: None) –
- lat (Optional[float], default: None) –
- long (Optional[float], default: None) –
- reply_to (Optional[int], default: None) –
- forward_messages (Optional[List[int]], default: None) –
- sticker_id (Optional[int], default: None) –
- template (Optional[Dict], default: None) –
- content_source (Optional[Dict], default: None) –
- dont_parse_links (Optional[int], default: None) –
- disable_mentions (Optional[int], default: None) –
- intent (str, default: 'default') –
- subscribe_id (Optional[int], default: None) –

type: str

class CallbackQuery

Описывает события типа MESSAGE_EVENT

user_id: int

peer_id: int

event_id: str

payload: DotDict

conversation_message_id: int

answer(*event_data=None*)

Параметры

event_data (Optional[dict], default: None) –

type: str

vk_maria.types.message_event

Classes

MessageEvent

vk_maria.types.message_event.MessageEvent

class MessageEvent(*vk*, *raw*)

Базовые классы: *Event*, *Message*

Methods

answer	**param message**
reply	Не работает в беседах из-за ограниченного апи

Attributes

from_group: bool

from_user: bool

message: *MessageInfo*

from_chat: bool

peer_id: int

type: str

chat_id: Optional[int]

answer(*message=None*, ***kwargs*)

Параметры
message (Optional[str], default: None) –

reply(*message=None*, ***kwargs*)

Не работает в беседах из-за ограниченного апи

Параметры
message (Optional[str], default: None) –

class MessageEvent(*vk*, *raw*)

from_group: bool

from_user: bool

message: *MessageInfo*

from_chat: bool

peer_id: int

type: str

chat_id: Optional[int]

answer(*message=None*, ***kwargs*)

Параметры
message (Optional[str], default: None) –

reply(*message=None*, ***kwargs*)

Не работает в беседах из-за ограниченного апи

Параметры
message (Optional[str], default: None) –

vk_maria.types.state

vk_maria.upload

vk_maria.upload.exceptions
vk_maria.upload.upload
vk_maria.upload.utils

vk_maria.upload.exceptions

Exceptions

InvalidFileFormatError

vk_maria.upload.exceptions.InvalidFileFormatError

exception InvalidFileFormatError

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception InvalidFileFormatError

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

vk_maria.upload.upload

Classes

Upload	Класс реализующий загрузку файлов на сервер вк.

vk_maria.upload.upload.Upload

class Upload(*vk*)

Базовые классы: object

Класс реализующий загрузку файлов на сервер вк.

Параметры
vk (*Vk*) –

Methods

document	Загрузка документа, возвращает объект для вставки в сообщение
photo	Загрузка фотографии, возвращает объект для вставки в сообщение
set_chat_photo	Установка обложки чата
set_group_cover_photo	Загрузка и установка обложки сообщества

photo(*photo*)

Загрузка фотографии, возвращает объект для вставки в сообщение

Параметры
photo (Union[str, bytes, PathLike]) –

set_chat_photo(*photo*, *chat_id*, *crop_x=None*, *crop_y=None*, *crop_width=None*)

Установка обложки чата

Параметры

- photo (Union[str, bytes, PathLike]) –
- chat_id (int) –
- crop_x (Optional[int], default: None) –
- crop_y (Optional[int], default: None) –
- crop_width (Optional[int], default: None) –

set_group_cover_photo(*photo*, *crop_x=None*, *crop_y=None*, *crop_x2=None*, *crop_y2=None*)

Загрузка и установка обложки сообщества

Параметры

- photo (Union[str, bytes, PathLike]) –
- crop_x (Optional[int], default: None) –
- crop_y (Optional[int], default: None) –
- crop_x2 (Optional[int], default: None) –
- crop_y2 (Optional[int], default: None) –

document(*document*, *peer_id*, *title=None*, *tags=None*, *return_tags=None*, *type='doc'*)

Загрузка документа, возвращает объект для вставки в сообщение

Параметры

- document (Union[str, BinaryIO, PathLike, List[Union[str, BinaryIO, PathLike]]]) –
- peer_id (int) –
- title (Optional[str], default: None) –
- tags (Optional[List[str]], default: None) –
- return_tags (Optional[int], default: None) –
- type (str, default: 'doc') –

class Upload(*vk*)

Класс реализующий загрузку файлов на сервер вк.

Параметры
vk (*Vk*) –

photo(*photo*)

Загрузка фотографии, возвращает объект для вставки в сообщение

Параметры
photo (Union[str, bytes, PathLike]) –

set_chat_photo(*photo*, *chat_id*, *crop_x=None*, *crop_y=None*, *crop_width=None*)

Установка обложки чата

Параметры

- photo (Union[str, bytes, PathLike]) –
- chat_id (int) –
- crop_x (Optional[int], default: None) –
- crop_y (Optional[int], default: None) –
- crop_width (Optional[int], default: None) –

set_group_cover_photo(*photo*, *crop_x=None*, *crop_y=None*, *crop_x2=None*, *crop_y2=None*)

Загрузка и установка обложки сообщества

Параметры

- photo (Union[str, bytes, PathLike]) –
- crop_x (Optional[int], default: None) –
- crop_y (Optional[int], default: None) –
- crop_x2 (Optional[int], default: None) –
- crop_y2 (Optional[int], default: None) –

document(*document*, *peer_id*, *title=None*, *tags=None*, *return_tags=None*, *type='doc'*)

Загрузка документа, возвращает объект для вставки в сообщение

Параметры

- document (Union[str, BinaryIO, PathLike, List[Union[str, BinaryIO, PathLike]]]) –
- peer_id (int) –
- title (Optional[str], default: None) –

- tags (Optional[List[str]], default: None) –
- return_tags (Optional[int], default: None) –
- type (str, default: 'doc') –

vk_maria.upload.utils

Functions

open_file	**param file**
open_files	**param files**

vk_maria.upload.utils.open_file

open_file(*file*)

Параметры
file (Union[str, BinaryIO]) –

vk_maria.upload.utils.open_files

open_files(*files*, *type*)

Параметры

- files (Union[str, BinaryIO, List[Union[str, BinaryIO]]]) –
- type (str) –

open_file(*file*)

Параметры
file (Union[str, BinaryIO]) –

open_files(*files*, *type*)

Параметры

- files (Union[str, BinaryIO, List[Union[str, BinaryIO]]]) –
- type (str) –

vk_maria.utils

Functions

args_converter
error_catcher
get_random_id
query_delimiter
response_parser

vk_maria.utils.args_converter

args_converter(*method*)

vk_maria.utils.error_catcher

error_catcher(*method*)

vk_maria.utils.get_random_id

get_random_id()

vk_maria.utils.query_delimiter

query_delimiter(*method*)

vk_maria.utils.response_parser

response_parser(*method*)

error_catcher(*method*)

query_delimiter(*method*)

get_random_id()

response_parser(*method*)

args_converter(*method*)

vk_maria.vk_types

Classes

Adresses
BanInfo
Career
City
Contacts
Counters
Country
Cover
Currency
Document
Education
Group
Images
LastSeen
Links
Market
Military
Occupation
Order
Personal
Place
Profile
Relatives
Schools
Universities

vk_maria.vk_types.Adresses

class Adresses(*args, **kwargs)

Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

is_enabled
main_address_id

is_enabled: bool

main_address_id: int

clear() → None. Remove all items from D.

copy() → a shallow copy of D

`fromkeys(value=None, /)`
Create a new dictionary with keys from iterable and values set to value.

`get(key, default=None, /)`
Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

`keys()` → a set-like object providing a view on D's keys

`pop(k[, d])` → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

`popitem()`
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

`setdefault(key, default=None, /)`
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

`update([E], **F)` → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

`values()` → an object providing a view on D's values

vk_maria.vk_types.BanInfo

`class BanInfo`
Базовые классы: `object`

Methods

Attributes

admin_id
date
reason
comment
end_date

admin_id: int

date: int

reason: int

comment: str

end_date: int

vk_maria.vk_types.Career

class Career(*args, **kwargs)

Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

group_id
company
country_id
city_id
city_name
*from_*
until
position

`group_id: int`

`company: str`

`country_id: int`

`city_id: int`

`city_name: str`

`from_: int`

`until: int`

`position: str`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`
Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`
Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

`keys()` → a set-like object providing a view on D's keys

`pop(`$k\left[, d\right]$`)` → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

`popitem()`
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

`setdefault(`*key*, *default=None*, */*`)`

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

`update(`$\left[E\right]$, ***F*`)` → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

`values()` → an object providing a view on D's values

vk_maria.vk_types.City

`class City(`**args*, ***kwargs*`)`

Базовые классы: `dict`

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

id
title

id: int

title: str

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

vk_maria.vk_types.Contacts

class Contacts(**args*, ***kwargs*)

Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

mobile_phone
home_phone

`user_id: int`

`desc: str`

`phone: str`

`email: str`

`mobile_phone: str`

`home_phone: str`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(value=None, /)`
Create a new dictionary with keys from iterable and values set to value.

`get(key, default=None, /)`
Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

`keys()` → a set-like object providing a view on D's keys

`pop(k[, d])` → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

`popitem()`
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

`setdefault(key, default=None, /)`
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

`update([E], **F)` → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

`values()` → an object providing a view on D's values

vk_maria.vk_types.Counters

`class Counters(*args, **kwargs)`
Базовые классы: `dict`

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

albums
videos
audios
photos
notes
friends
groups
online_friends
mutual_friends
user_videos
followers
pages

albums: int

videos: int

audios: int

photos: int

notes: int

friends: int

groups: int

online_friends: int

mutual_friends: int

user_videos: int

followers: int

pages: int

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, */*)
: Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)
: Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
: If key is not found, d is returned if given, otherwise KeyError is raised

popitem()
: Remove and return a (key, value) pair as a 2-tuple.

  Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)
: Insert key with a value of default if key is not in the dictionary.

  Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
: If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

vk_maria.vk_types.Country

class Country(**args*, ***kwargs*)
: Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

id	
title	

`id: int`

`title: str`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`

Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`

Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem()
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

vk_maria.vk_types.Cover

class Cover(**args*, ***kwargs*)
Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

enabled	
images	

`enabled: int`

`images: List[`*Images*`]`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`
Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`
Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem()
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

vk_maria.vk_types.Currency

class Currency(**args*, ***kwargs*)
Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

id	
name	

`id: int`

`name: str`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`

Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`

Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem()
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

vk_maria.vk_types.Document

class Document
Базовые классы: object

Methods

Attributes

id
owner_id
title
size
ext
url
date
type
preview

id: int

owner_id: int

title: str

size: int

ext: str

url: str

date: int

type: int

preview: Dict

vk_maria.vk_types.Education

class Education(*args, **kwargs)

Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

university
university_name
faculty
faculty_name
graduation

`university: int`

`university_name: str`

`faculty: int`

`faculty_name: str`

`graduation: int`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`
Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`
Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

`keys()` → a set-like object providing a view on D's keys

`pop(`$k\left[, d\right]$`)` → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

`popitem()`
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

`setdefault(`*key*, *default=None*, */*`)`
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

`update(`$\left[E\right]$, ***F*`)` → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

`values()` → an object providing a view on D's values

vk_maria.vk_types.Group

class Group

Базовые классы: object

Methods

Attributes

activity
addresses
age_limits
ban_info
can_create_topic
can_message
can_post
can_see_all_posts
can_upload_doc
can_upload_video
city
contacts
counters
country
cover
crop_photo
description
finish_date
fixed_post

continues on next page

Таблица 6 – продолжение с предыдущей страницы

has_photo
is_favorite
is_hidden_from_feed
is_messages_blocked
links
main_album_id
main_section
market
member_status
members_count
place
public_date_label
site
start_date
status
trending
verified
wall
wiki_page
id
name
screen_name
is_closed
deactivated
is_admin
admin_level

continues on next page

Таблица 6 – продолжение с предыдущей страницы

is_member
is_advertiser
invited_by
type
photo_50
photo_100
photo_200

```
id: int
name: str
screen_name: str
is_closed: int
deactivated: str
is_admin: int
admin_level: int
is_member: int
is_advertiser: int
invited_by: int
type: str
photo_50: str
photo_100: str
photo_200: str
activity: str = None
addresses: Adresses = None
age_limits: int = None
ban_info: BanInfo = None
can_create_topic: int = None
can_message: int = None
can_post: int = None
```

```
    can_see_all_posts: int = None
    can_upload_doc: int = None
    can_upload_video: int = None
    city: City = None
    contacts: Contacts = None
    counters: Counters = None
    country: Country = None
    cover: Cover = None
    crop_photo: Dict = None
    description: str = None
    fixed_post: int = None
    has_photo: int = None
    is_favorite: int = None
    is_hidden_from_feed: int = None
    is_messages_blocked: int = None
    links: Links = None
    main_album_id: int = None
    main_section: int = None
    market: Market = None
    member_status: int = None
    members_count: int = None
    place: Place = None
    public_date_label: str = None
    site: str = None
    start_date: int = None
    finish_date: int = None
    status: str = None
    trending: int = None
    verified: int = None
    wall: int = None
    wiki_page: str = None
```

vk_maria.vk_types.Images

class Images(*args, **kwargs)

Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

url
width
height

url: str

width: int

height: int

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, /)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, /)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, /)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

vk_maria.vk_types.LastSeen

class LastSeen(**args*, ***kwargs*)

Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

time	
platform	

`time: int`

`platform: int`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`

Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`

Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem()
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

vk_maria.vk_types.Links

class Links(**args*, ***kwargs*)
Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

id
url
name
desc
photo_50
photo_100

```
id: int

url: str

name: str

desc: str
```

photo_50: str

photo_100: str

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, /)
Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, /)
Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem()
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, /)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

vk_maria.vk_types.Market

class Market(**args*, ***kwargs*)
Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

enabled
type
price_min
price_max
main_album_id
contact_id
currency
currency_text

`enabled: int`

`type: str`

price_min: int

price_max: int

main_album_id: int

contact_id: int

currency: *Currency*

currency_text: str

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

vk_maria.vk_types.Military

class Military(**args*, ***kwargs*)

Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

unit
unit_id
country_id
*from_*
until

```
unit: str

unit_id: int

country_id: int

from_: int

until: int
```

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, /)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, /)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, /)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

vk_maria.vk_types.Occupation

class Occupation(**args*, ***kwargs*)

Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

type
id
name

`type: str`

`id: int`

`name: str`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`

Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`
Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

`keys()` → a set-like object providing a view on D's keys

`pop(`*k*[, *d*]`)` → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

`popitem()`
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

`setdefault(`*key*, *default=None*, */*`)`
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

`update(`[*E*], ***F*`)` → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

`values()` → an object providing a view on D's values

vk_maria.vk_types.Order

`class Order`
Базовые классы: `object`

Methods

Attributes

id
group_id
user_id
date
variants_grouping_id
is_main_variant
property_values
cart_quantity
status
items_count
total_price
display_order_id
comment
preview_order_items
delivery
recipient

id: int

group_id: int

user_id: int

date: int

variants_grouping_id: int

is_main_variant: bool

property_values: List

cart_quantity: int

status: int

items_count: int

total_price: Dict

display_order_id: str

comment: str

preview_order_items: List

delivery: Dict

recipient: Dict

vk_maria.vk_types.Personal

class Personal(*args, **kwargs)

Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

political
langs
religion
inspired_by
people_main
life_main
smoking
alcohol

`political: int`

`langs: List[str]`

`religion: str`

`inspired_by: str`

`people_main: int`

`life_main: int`

`smoking: int`

`alcohol: int`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`
Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`
Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

`keys()` → a set-like object providing a view on D's keys

`pop(`$k\left[, d\right]$`)` → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

`popitem()`
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

`setdefault(`*key*, *default=None*, */*`)`

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

`update(`$\left[E\right]$, ***F*`)` → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

`values()` → an object providing a view on D's values

vk_maria.vk_types.Place

`class Place(`**args*, ***kwargs*`)`

Базовые классы: `dict`

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

id
title
latitude
longitude
type
country
city
address

`id: int`

`title: str`

`latitude: float`

`longitude: float`

`type: str`

`country: int`

`city: int`

`address: str`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`
Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`
Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

`keys()` → a set-like object providing a view on D's keys

`pop(`$k\left[, d\right]$`)` → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

`popitem()`
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update(*[E]*, ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

vk_maria.vk_types.Profile

class Profile

Базовые классы: object

Methods

Attributes

about
activities
bdate
blacklisted
blacklisted_by_me
books
can_post
can_see_all_posts
can_see_audio
can_send_friend_request
can_write_private_message
career
city

continues on next page

Таблица 7 – продолжение с предыдущей страницы

common_count
connections
contacts
counters
country
crop_photo
domain
education
exports
first_name_abl
first_name_acc
first_name_dat
first_name_gen
first_name_ins
first_name_nom
followers_count
friend_status
games
has_mobile
has_photo
home_town
interests
is_favorite
is_friend
is_hidden_from_feed
last_name_abl

continues on next page

Таблица 7 – продолжение с предыдущей страницы

last_name_acc
last_name_dat
last_name_gen
last_name_ins
last_name_nom
last_seen
lists
maiden_name
military
movies
music
nickname
occupation
online
personal
photo_100
photo_200
photo_200_orig
photo_400_orig
photo_50
photo_id
photo_max
photo_max_orig
quotes
relation
relatives

continues on next page

Таблица 7 – продолжение с предыдущей страницы

schools
screen_name
sex
site
status
timezone
trending
tv
universities
verified
wall_default
id
first_name
last_name
deactivated
is_closed
can_access_closed

```
id: int

first_name: str

last_name: str

deactivated: str

is_closed: bool

can_access_closed: bool

about: str = None

activities: str = None

bdate: str = None

blacklisted: int = None
```

```
    blacklisted_by_me: int = None

    books: str = None

    can_post: int = None

    can_see_all_posts: int = None

    can_see_audio: int = None

    can_send_friend_request: int = None

    can_write_private_message: int = None

    career: Career = None

    city: City = None

    common_count: int = None

    connections: str = None

    contacts: Contacts = None

    counters: Counters = None

    country: Country = None

    crop_photo: dict = None

    domain: str = None

    education: Education = None

    exports = None

    first_name_nom: str = None

    first_name_gen: str = None

    first_name_dat: str = None

    first_name_acc: str = None

    first_name_ins: str = None

    first_name_abl: str = None

    followers_count: int = None

    friend_status: int = None

    games: str = None

    has_mobile: int = None

    has_photo: int = None

    home_town: str = None

    interests: str = None
```

```
is_favorite: int = None
is_friend: int = None
is_hidden_from_feed: int = None
last_name_nom: str = None
last_name_gen: str = None
last_name_dat: str = None
last_name_acc: str = None
last_name_ins: str = None
last_name_abl: str = None
last_seen: LastSeen = None
lists: str = None
maiden_name: str = None
military: Military = None
movies: str = None
music: str = None
nickname: str = None
occupation: Occupation = None
online: int = None
personal: Personal = None
photo_50: str = None
photo_100: str = None
photo_200_orig: str = None
photo_200: str = None
photo_400_orig: str = None
photo_id: str = None
photo_max: str = None
photo_max_orig: str = None
quotes: str = None
relatives: Relatives = None
relation: int = None
schools: Schools = None
```

screen_name: str = None

sex: int = None

site: str = None

status: str = None

timezone: int = None

trending: int = None

tv: str = None

universities: *Universities* = None

verified: int = None

wall_default: str = None

vk_maria.vk_types.Relatives

class Relatives(**args*, ***kwargs*)

Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

id
name
type

`id: int`

`name: str`

`type: str`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`
Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`
Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

`keys()` → a set-like object providing a view on D's keys

`pop(`$k\left[, d\right]$`)` → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

`popitem()`
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

`setdefault(`*key*, *default=None*, */*`)`
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

`update(`$\left[E\right]$, ***F*`)` → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

`values()` → an object providing a view on D's values

vk_maria.vk_types.Schools

class Schools(**args*, ***kwargs*)

Базовые классы: dict

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

id
country
city
name
year_from
year_to
year_graduated
*class_*
speciality
type
type_str

id: int

country: int

city: int

name: str

year_from: int

year_to: int

year_graduated: int

class_: str

speciality: str

type: int

type_str: str

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, */*)
: Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)
: Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

`keys()` → a set-like object providing a view on D's keys

`pop(`*k*[, *d*]`)` → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

`popitem()`
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

`setdefault(`*key*, *default=None*, */*`)`
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

`update(`[*E*], ***F*`)` → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

`values()` → an object providing a view on D's values

vk_maria.vk_types.Universities

`class Universities(`**args*, ***kwargs*`)`
Базовые классы: `dict`

Methods

clear	
copy	
fromkeys	Create a new dictionary with keys from iterable and values set to value.
get	Return the value for key if key is in the dictionary, else default.
items	
keys	
pop	If key is not found, d is returned if given, otherwise KeyError is raised
popitem	Remove and return a (key, value) pair as a 2-tuple.
setdefault	Insert key with a value of default if key is not in the dictionary.
update	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values	

Attributes

id
country
city
name
faculty
faculty_name
chair
chair_name
graduation
education_form
education_status

`id: int`

`country: int`

`city: int`

`name: str`

`faculty: int`

`faculty_name: str`

`chair: int`

`chair_name: str`

`graduation: int`

`education_form: str`

`education_status: str`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`
Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`
Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem()
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Adresses(**args*, ***kwargs*)

is_enabled: bool

main_address_id: int

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, */*)
Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)
Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem()
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Images(**args*, ***kwargs*)

url: str

width: int

height: int

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Cover(**args*, ***kwargs*)

enabled: int

images: List[*Images*]

clear() → None. Remove all items from D.

copy() → a shallow copy of D

`fromkeys(`*value=None*, */*`)`

Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`

Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

`keys()` → a set-like object providing a view on D's keys

`pop(`*k*[, *d*]`)` → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

`popitem()`

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

`setdefault(`*key*, *default=None*, */*`)`

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

`update(`[*E*], ***F*`)` → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

`values()` → an object providing a view on D's values

`class Links(`**args*, ***kwargs*`)`

`id: int`

`url: str`

`name: str`

`desc: str`

`photo_50: str`

`photo_100: str`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`

Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`

Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

`keys()` → a set-like object providing a view on D's keys

`pop(`*k*[, *d*]`)` → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

`popitem()`
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

`setdefault(`*key*, *default=None*, */*`)`
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

`update(`[*E*], ***F*`)` → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

`values()` → an object providing a view on D's values

`class Currency(`**args*, ***kwargs*`)`

`id: int`

`name: str`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`
Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`
Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

`keys()` → a set-like object providing a view on D's keys

`pop(`*k*[, *d*]`)` → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

`popitem()`
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

`setdefault(`*key*, *default=None*, */*`)`
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

`update(`[*E*], ***F*`)` → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

`values()` → an object providing a view on D's values

class Market(*args, **kwargs)

enabled: int

type: str

price_min: int

price_max: int

main_album_id: int

contact_id: int

currency: *Currency*

currency_text: str

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, /)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, /)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, /)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Place(*args, **kwargs)

id: int

title: str

latitude: float

longitude: float

type: str

country: int

city: int

address: str

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, /)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, /)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, /)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Group

id: int

name: str

screen_name: str

is_closed: int

deactivated: str

is_admin: int

admin_level: int

```
is_member: int
is_advertiser: int
invited_by: int
type: str
photo_50: str
photo_100: str
photo_200: str
activity: str = None
addresses: Adresses = None
age_limits: int = None
ban_info: BanInfo = None
can_create_topic: int = None
can_message: int = None
can_post: int = None
can_see_all_posts: int = None
can_upload_doc: int = None
can_upload_video: int = None
city: City = None
contacts: Contacts = None
counters: Counters = None
country: Country = None
cover: Cover = None
crop_photo: Dict = None
description: str = None
fixed_post: int = None
has_photo: int = None
is_favorite: int = None
is_hidden_from_feed: int = None
is_messages_blocked: int = None
links: Links = None
main_album_id: int = None
```

main_section: int = None

market: *Market* = None

member_status: int = None

members_count: int = None

place: *Place* = None

public_date_label: str = None

site: str = None

start_date: int = None

finish_date: int = None

status: str = None

trending: int = None

verified: int = None

wall: int = None

wiki_page: str = None

class Career(**args*, ***kwargs*)

group_id: int

company: str

country_id: int

city_id: int

city_name: str

from_: int

until: int

position: str

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem()
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class City(**args*, ***kwargs*)

id: int

title: str

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, */*)
Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)
Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem()
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Contacts(**args*, ***kwargs*)

mobile_phone: str

home_phone: str

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Counters(**args*, ***kwargs*)

albums: int

videos: int

audios: int

photos: int

notes: int

friends: int

groups: int

online_friends: int

mutual_friends: int

user_videos: int

followers: int

pages: int

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, /)
Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, /)
Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem()
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, /)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Country(**args*, ***kwargs*)

id: int

title: str

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, /)
Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, /)
Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem()
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, /)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Education(**args*, ***kwargs*)

university: int

university_name: str

faculty: int

faculty_name: str

graduation: int

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, /)
Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, /)
Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem()
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, /)
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update(*E*, ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class LastSeen(**args*, ***kwargs*)

time: int

platform: int

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*, *d*) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update(*E*, ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Military(**args*, ***kwargs*)

unit: str

unit_id: int

country_id: int

from_: int

until: int

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`
Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`
Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

`keys()` → a set-like object providing a view on D's keys

`pop(`$k\left[, d\right]$`)` → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

`popitem()`
Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

`setdefault(`*key*, *default=None*, */*`)`
Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

`update(`$\left[E\right]$, ***F*`)` → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

`values()` → an object providing a view on D's values

`class Occupation(`**args*, ***kwargs*`)`

`type: str`

`id: int`

`name: str`

`clear()` → None. Remove all items from D.

`copy()` → a shallow copy of D

`fromkeys(`*value=None*, */*`)`
Create a new dictionary with keys from iterable and values set to value.

`get(`*key*, *default=None*, */*`)`
Return the value for key if key is in the dictionary, else default.

`items()` → a set-like object providing a view on D's items

`keys()` → a set-like object providing a view on D's keys

`pop(`$k\left[, d\right]$`)` → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Personal(**args*, ***kwargs*)

political: int

langs: List[str]

religion: str

inspired_by: str

people_main: int

life_main: int

smoking: int

alcohol: int

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update(*E*, **F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Relatives(*args, **kwargs)

id: int

name: str

type: str

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, /)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, /)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*, *d*) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, /)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update(*E*, **F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Schools(*args, **kwargs)

id: int

country: int

city: int

name: str

year_from: int

year_to: int

year_graduated: int

class_: str

speciality: str

type: int

type_str: str

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Universities(**args*, ***kwargs*)

id: int

country: int

city: int

name: str

faculty: int

faculty_name: str

chair: int

chair_name: str

graduation: int

education_form: str

education_status: str

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None*, */*)

Create a new dictionary with keys from iterable and values set to value.

get(*key*, *default=None*, */*)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises KeyError if the dict is empty.

setdefault(*key*, *default=None*, */*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class Profile

id: int

first_name: str

last_name: str

deactivated: str

is_closed: bool

can_access_closed: bool

```
about: str = None
activities: str = None
bdate: str = None
blacklisted: int = None
blacklisted_by_me: int = None
books: str = None
can_post: int = None
can_see_all_posts: int = None
can_see_audio: int = None
can_send_friend_request: int = None
can_write_private_message: int = None
career: Career = None
city: City = None
common_count: int = None
connections: str = None
contacts: Contacts = None
counters: Counters = None
country: Country = None
crop_photo: dict = None
domain: str = None
education: Education = None
exports = None
first_name_nom: str = None
first_name_gen: str = None
first_name_dat: str = None
first_name_acc: str = None
first_name_ins: str = None
first_name_abl: str = None
followers_count: int = None
friend_status: int = None
games: str = None
```

```
has_mobile: int = None

has_photo: int = None

home_town: str = None

interests: str = None

is_favorite: int = None

is_friend: int = None

is_hidden_from_feed: int = None

last_name_nom: str = None

last_name_gen: str = None

last_name_dat: str = None

last_name_acc: str = None

last_name_ins: str = None

last_name_abl: str = None

last_seen: LastSeen = None

lists: str = None

maiden_name: str = None

military: Military = None

movies: str = None

music: str = None

nickname: str = None

occupation: Occupation = None

online: int = None

personal: Personal = None

photo_50: str = None

photo_100: str = None

photo_200_orig: str = None

photo_200: str = None

photo_400_orig: str = None

photo_id: str = None

photo_max: str = None

photo_max_orig: str = None
```

```
    quotes: str = None
    relatives: Relatives = None
    relation: int = None
    schools: Schools = None
    screen_name: str = None
    sex: int = None
    site: str = None
    status: str = None
    timezone: int = None
    trending: int = None
    tv: str = None
    universities: Universities = None
    verified: int = None
    wall_default: str = None
class BanInfo
    admin_id: int
    date: int
    reason: int
    comment: str
    end_date: int
class Order
    id: int
    group_id: int
    user_id: int
    date: int
    variants_grouping_id: int
    is_main_variant: bool
    property_values: List
    cart_quantity: int
    status: int
    items_count: int
```

```
    total_price: Dict
    display_order_id: str
    comment: str
    preview_order_items: List
    delivery: Dict
    recipient: Dict
class Document
    id: int
    owner_id: int
    title: str
    size: int
    ext: str
    url: str
    date: int
    type: int
    preview: Dict
```

Глава 4

Indices and tables

- genindex
- modindex
- search

Содержание модулей Python

V

vk_maria, 14
vk_maria.api, 15
vk_maria.dispatcher, 64
vk_maria.dispatcher.dispatcher, 64
vk_maria.dispatcher.filters, 67
vk_maria.dispatcher.filters.filters, 67
vk_maria.dispatcher.filters.handler, 78
vk_maria.dispatcher.fsm, 79
vk_maria.dispatcher.fsm.state, 80
vk_maria.dispatcher.fsm.storage, 82
vk_maria.dispatcher.fsm.storage.core, 82
vk_maria.dispatcher.fsm.storage.file, 92
vk_maria.dispatcher.fsm.storage.file.base, 92
vk_maria.dispatcher.fsm.storage.file.json, 97
vk_maria.dispatcher.fsm.storage.file.pickle, 102
vk_maria.dispatcher.fsm.storage.memory, 107
vk_maria.dispatcher.fsm.storage.memory.memory, 107
vk_maria.exceptions, 112
vk_maria.longpoll, 116
vk_maria.longpoll.longpoll, 116
vk_maria.mixins, 117
vk_maria.responses, 117
vk_maria.types, 159
vk_maria.types.callback_query_event, 159
vk_maria.types.chat, 160
vk_maria.types.event, 162
vk_maria.types.event_type, 162
vk_maria.types.keyboard, 167
vk_maria.types.message, 178
vk_maria.types.message_event, 187
vk_maria.types.state, 189
vk_maria.upload, 189
vk_maria.upload.exceptions, 189
vk_maria.upload.upload, 189
vk_maria.upload.utils, 192
vk_maria.utils, 193
vk_maria.vk_types, 194

Символы

модуль
vk_maria, 14
vk_maria.api, 15
vk_maria.dispatcher, 64
vk_maria.dispatcher.dispatcher, 64
vk_maria.dispatcher.filters, 67
vk_maria.dispatcher.filters.filters, 67
vk_maria.dispatcher.filters.handler, 78
vk_maria.dispatcher.fsm, 79
vk_maria.dispatcher.fsm.state, 80
vk_maria.dispatcher.fsm.storage, 82
vk_maria.dispatcher.fsm.storage.core, 82
vk_maria.dispatcher.fsm.storage.file, 92
vk_maria.dispatcher.fsm.storage.file.base, 92
vk_maria.dispatcher.fsm.storage.file.json, 97
vk_maria.dispatcher.fsm.storage.file.pickle, 102
vk_maria.dispatcher.fsm.storage.memory, 107
vk_maria.dispatcher.fsm.storage.memory.memory, 107
vk_maria.exceptions, 112
vk_maria.longpoll, 116
vk_maria.longpoll.longpoll, 116
vk_maria.mixins, 117
vk_maria.responses, 117
vk_maria.types, 159
vk_maria.types.callback_query_event, 159
vk_maria.types.chat, 160
vk_maria.types.event, 162
vk_maria.types.event_type, 162
vk_maria.types.keyboard, 167
vk_maria.types.message, 178
vk_maria.types.message_event, 187
vk_maria.types.state, 189
vk_maria.upload, 189
vk_maria.upload.exceptions, 189
vk_maria.upload.upload, 189
vk_maria.upload.utils, 192
vk_maria.utils, 193
vk_maria.vk_types, 194
__call__() (*метод ApiMethod*), 15
__call__() (*метод BoundFilterMeta*), 69
__call__() (*метод HandlerObject*), 79
__call__() (*метод KeyboardModelMeta*), 172
__call__() (*метод StatesGroupMeta*), 81

A

about (*ампибут Profile*), 239, 266
AbstractFilter (*класс в vk_maria.dispatcher.filters.filters*), 68, 75
access_key (*ампибут UtilsGetShortLink*), 150, 158
AccessIsDeniedError, 112, 115
action (*ампибут BaseButton*), 169, 175
action (*ампибут CallbackButton*), 170, 176
action (*ампибут LocationButton*), 173, 176
action (*ампибут OpenLinkButton*), 173, 176
action (*ампибут TextButton*), 174, 175
action (*ампибут VKAppsButton*), 175, 176
action (*ампибут VKPayButton*), 175, 176
activities (*ампибут Profile*), 239, 267
activity (*ампибут Group*), 217, 255
activity (*ампибут GroupsGetById*), 124, 151
add_button() (*метод KeyboardMarkup*), 171, 178
add_row() (*метод KeyboardMarkup*), 171, 178
address (*ампибут Place*), 235, 254
addresses (*ампибут Group*), 217, 255
addresses (*ампибут GroupsGetById*), 124, 151
admin_id (*ампибут BanInfo*), 197, 269
admin_level (*ампибут Group*), 217, 254
admin_level (*ампибут GroupsGetById*), 125, 153
Adresses (*класс в vk_maria.vk_types*), 196, 249
age_limits (*ампибут Group*), 217, 255
age_limits (*ампибут GroupsGetById*), 124, 151
aid (*ампибут PhotosSaveMessagesPhoto*), 142, 157

album_id (*атрибут PhotosGetMessagesUploadServer*), 140, 156
albums (*атрибут Counters*), 205, 258
alcohol (*атрибут Personal*), 233, 263
answer (*атрибут StoriesGetStats*), 147, 158
answer() (*метод CallbackQuery*), 179, 187
answer() (*метод CallbackQueryEvent*), 160
answer() (*метод Message*), 180, 185
answer() (*метод MessageEvent*), 188
ApiMethod (*класс в vk_maria.api*), 15, 43
args (*атрибут AccessIsDeniedError*), 112, 115
args (*атрибут AuthorizationError*), 112, 115
args (*атрибут DeprecatedMethodError*), 113, 115
args (*атрибут FiltersFactory.UnknownFilterException*), 72, 77
args (*атрибут InvalidFileFormatError*), 189
args (*атрибут InvalidParametersError*), 113, 115
args (*атрибут KeyIsNotValidError*), 113, 116
args (*атрибут PermissionError*), 113, 115
args (*атрибут ServerError*), 113, 115
args (*атрибут UnknownError*), 114, 115
args (*атрибут UnknownMethodError*), 114, 115
args (*атрибут VkMariaException*), 114, 116
args (*атрибут WrongRequestError*), 114, 115
args_converter() (*в модуле vk_maria.utils*), 193
attachments (*атрибут MessageInfo*), 183, 185
attachments (*атрибут MessagesPin*), 138, 155
AUDIO_NEW (*атрибут EventType*), 165, 166
audios (*атрибут Counters*), 205, 258
AuthorizationError, 112, 115

B

ban_info (*атрибут Group*), 217, 255
ban_info (*атрибут GroupsGetById*), 124, 151
BanInfo (*класс в vk_maria.vk_types*), 197, 269
bans (*атрибут StoriesGetStats*), 147, 158
base_url (*атрибут ApiMethod*), 15, 43
BaseButton (*класс в vk_maria.types.keyboard*), 168, 175
BaseEvent (*класс в vk_maria.types.message*), 178, 184
BaseStorage (*класс в vk_maria.dispatcher.fsm.storage.core*), 83, 88
bdate (*атрибут Profile*), 239, 267
blacklisted (*атрибут Profile*), 239, 267
blacklisted_by_me (*атрибут Profile*), 239, 267
board_delete_comment() (*метод Vk*), 36, 57
BOARD_POST_DELETE (*атрибут EventType*), 165, 167
BOARD_POST_EDIT (*атрибут EventType*), 165, 166
BOARD_POST_NEW (*атрибут EventType*), 165, 166
BOARD_POST_RESTORE (*атрибут EventType*), 165, 167
board_restore_comment() (*метод Vk*), 36, 57
books (*атрибут Profile*), 240, 267
BoundFilter (*класс в vk_maria.dispatcher.filters.filters*), 68, 75
BoundFilterMeta (*класс в vk_maria.dispatcher.filters.filters*), 69, 75
Button (*класс в vk_maria.types.keyboard*), 169, 176

C

Callback (*атрибут Button*), 169, 177
callback_handler() (*метод Dispatcher*), 66, 67
CallbackButton (*класс в vk_maria.types.keyboard*), 170, 176
CallbackQuery (*класс в vk_maria.types.message*), 179, 187
CallbackQueryEvent (*класс в vk_maria.types.callback_query_event*), 159, 160
can_access_closed (*атрибут Profile*), 239, 266
can_create_topic (*атрибут Group*), 217, 255
can_create_topic (*атрибут GroupsGetById*), 124, 151
can_invite (*атрибут GroupsIsMember*), 128, 153
can_message (*атрибут Group*), 217, 255
can_message (*атрибут GroupsGetById*), 124, 151
can_post (*атрибут Group*), 217, 255
can_post (*атрибут GroupsGetById*), 124, 151
can_post (*атрибут Profile*), 240, 267
can_recall (*атрибут GroupsIsMember*), 128, 153
can_see_all_posts (*атрибут Group*), 217, 255
can_see_all_posts (*атрибут GroupsGetById*), 124, 151
can_see_all_posts (*атрибут Profile*), 240, 267
can_see_audio (*атрибут Profile*), 240, 267
can_send_friend_request (*атрибут Profile*), 240, 267
can_upload_doc (*атрибут Group*), 218, 255
can_upload_doc (*атрибут GroupsGetById*), 124, 151
can_upload_video (*атрибут Group*), 218, 255
can_upload_video (*атрибут GroupsGetById*), 124, 151
can_write_private_message (*атрибут Profile*), 240, 267
career (*атрибут Profile*), 240, 267
Career (*класс в vk_maria.vk_types*), 198, 256
cart_quantity (*атрибут Order*), 231, 269
chair (*атрибут Universities*), 248, 266
chair_name (*атрибут Universities*), 248, 266
chat (*атрибут MessagesDeleteChatPhoto*), 129, 154
chat (*атрибут MessagesSetChatPhoto*), 139, 156
Chat (*класс в vk_maria.types.chat*), 161
chat_id (*атрибут Chat*), 161

chat_id (*атрибут Message*), 180, 185
chat_id (*атрибут MessageEvent*), 188
check() (*метод AbstractFilter*), 68, 75
check() (*метод BoundFilter*), 68, 75
check() (*метод CommandsFilter*), 70, 76
check() (*метод EventTypeFilter*), 70, 76
check() (*метод FSMStateFilter*), 71, 77
check() (*метод FunctionFilter*), 72, 76
check() (*метод PayloadFilter*), 73, 77
check() (*метод RegexpFilter*), 74, 77
check() (*метод TextFilter*), 74, 76
check() (*метод TypeFromFilter*), 75, 77
check_address() (*метод класса BaseStorage*), 83, 88
check_address() (*метод класса DisabledStorage*), 86, 91
check_address() (*метод класса FileStorage*), 94, 95
check_address() (*метод класса JSONStorage*), 98, 100
check_address() (*метод класса MemoryStorage*), 109, 111
check_address() (*метод класса PickleStorage*), 103, 105
check_all() (*метод Filters*), 71, 77
city (*атрибут Group*), 218, 255
city (*атрибут GroupsGetById*), 124, 151
city (*атрибут Place*), 235, 254
city (*атрибут Profile*), 240, 267
city (*атрибут Schools*), 245, 264
city (*атрибут Universities*), 248, 265
City (*класс в vk_maria.vk_types*), 200, 257
city_id (*атрибут Career*), 199, 256
city_name (*атрибут Career*), 199, 256
class_ (*атрибут Schools*), 245, 265
clear() (*метод Adresses*), 196, 249
clear() (*метод Career*), 199, 256
clear() (*метод City*), 201, 257
clear() (*метод Contacts*), 202, 258
clear() (*метод Counters*), 205, 259
clear() (*метод Country*), 207, 259
clear() (*метод Cover*), 209, 250
clear() (*метод Currency*), 211, 252
clear() (*метод Education*), 214, 260
clear() (*метод Images*), 219, 250
clear() (*метод LastSeen*), 221, 261
clear() (*метод Links*), 224, 251
clear() (*метод Market*), 226, 253
clear() (*метод MessageInfo*), 183, 185
clear() (*метод Military*), 227, 261
clear() (*метод Occupation*), 229, 262
clear() (*метод Personal*), 233, 263
clear() (*метод Place*), 235, 254
clear() (*метод Relatives*), 243, 264
clear() (*метод Schools*), 245, 265
clear() (*метод Universities*), 248, 266
close() (*метод BaseStorage*), 83, 88
close() (*метод DisabledStorage*), 85, 90
close() (*метод FileStorage*), 93, 95
close() (*метод JSONStorage*), 99, 100
close() (*метод MemoryStorage*), 108, 110
close() (*метод PickleStorage*), 104, 105
Color (*класс в vk_maria.types.keyboard*), 170, 177
CommandsFilter (*класс в vk_maria.dispatcher.filters.filters*), 69, 76
comment (*атрибут BanInfo*), 198, 269
comment (*атрибут Order*), 232, 270
comment_id (*атрибут WallCreateComment*), 151, 159
common_count (*атрибут Profile*), 240, 267
company (*атрибут Career*), 199, 256
connections (*атрибут Profile*), 240, 267
construct_json() (*в модуле vk_maria.types.keyboard*), 167, 177
contact_id (*атрибут Market*), 226, 253
contacts (*атрибут Group*), 218, 255
contacts (*атрибут GroupsGetById*), 124, 151
contacts (*атрибут Profile*), 240, 267
Contacts (*класс в vk_maria.vk_types*), 201, 257
conversation_message_id (*атрибут CallbackQuery*), 179, 187
conversation_message_id (*атрибут CallbackQueryEvent*), 160
conversation_message_id (*атрибут MessageInfo*), 183, 184
copy() (*метод Adresses*), 196, 249
copy() (*метод Career*), 199, 256
copy() (*метод City*), 201, 257
copy() (*метод Contacts*), 202, 258
copy() (*метод Counters*), 205, 259
copy() (*метод Country*), 207, 259
copy() (*метод Cover*), 209, 250
copy() (*метод Currency*), 211, 252
copy() (*метод Education*), 214, 260
copy() (*метод Images*), 219, 250
copy() (*метод LastSeen*), 221, 261
copy() (*метод Links*), 224, 251
copy() (*метод Market*), 226, 253
copy() (*метод MessageInfo*), 183, 185
copy() (*метод Military*), 228, 262
copy() (*метод Occupation*), 229, 262
copy() (*метод Personal*), 233, 263
copy() (*метод Place*), 235, 254
copy() (*метод Relatives*), 243, 264
copy() (*метод Schools*), 245, 265
copy() (*метод Universities*), 248, 266
count (*атрибут DocsSearch*), 121, 153

count (*атрибут GroupsGetBanned*), 122, 151
count (*атрибут GroupsGetMembers*), 126, 153
count (*атрибут MarketGetGroupOrders*), 128, 156
count (*атрибут MarketGetOrderItems*), 129, 156
count (*атрибут MessagesGetByConversationMessageId*), 130, 154
count (*атрибут MessagesGetById*), 131, 154
count (*атрибут MessagesGetConversationMembers*), 131, 154
count (*атрибут MessagesGetConversations*), 132, 154
count (*атрибут MessagesGetConversationsById*), 133, 155
count (*атрибут MessagesGetHistory*), 133, 155
count (*атрибут MessagesGetIntentUsers*), 135, 155
count (*атрибут MessagesSearch*), 138, 156
count (*атрибут MessagesSearchConversations*), 139, 156
count (*атрибут StoriesGet*), 144, 157
count (*атрибут StoriesGetById*), 145, 157
count (*атрибут StoriesGetReplies*), 146, 157
count (*атрибут StoriesGetViewers*), 148, 158
count (*атрибут StoriesSave*), 148, 158
counters (*атрибут Group*), 218, 255
counters (*атрибут GroupsGetById*), 124, 152
counters (*атрибут Profile*), 240, 267
Counters (*класс в vk_maria.vk_types*), 203, 258
country (*атрибут Group*), 218, 255
country (*атрибут GroupsGetById*), 124, 152
country (*атрибут Place*), 235, 254
country (*атрибут Profile*), 240, 267
country (*атрибут Schools*), 245, 264
country (*атрибут Universities*), 248, 265
Country (*класс в vk_maria.vk_types*), 206, 259
country_id (*атрибут Career*), 199, 256
country_id (*атрибут Military*), 227, 261
cover (*атрибут Group*), 218, 255
cover (*атрибут GroupsGetById*), 124, 152
Cover (*класс в vk_maria.vk_types*), 208, 250
created (*атрибут PhotosSaveMessagesPhoto*), 142, 157
crop_photo (*атрибут Group*), 218, 255
crop_photo (*атрибут GroupsGetById*), 124, 152
crop_photo (*атрибут Profile*), 240, 267
currency (*атрибут Market*), 226, 253
Currency (*класс в vk_maria.vk_types*), 210, 252
currency_text (*атрибут Market*), 226, 253

D

date (*атрибут BanInfo*), 198, 269
date (*атрибут DocsSave*), 121, 154
date (*атрибут Document*), 213, 270
date (*атрибут MessageInfo*), 183, 184
date (*атрибут MessagesPin*), 138, 155
date (*атрибут Order*), 231, 269
deactivated (*атрибут Group*), 217, 254
deactivated (*атрибут GroupsGetById*), 125, 152
deactivated (*атрибут Profile*), 239, 266
delivery (*атрибут Order*), 232, 270
DeprecatedMethodError, 113, 115
desc (*атрибут Contacts*), 202
desc (*атрибут Links*), 223, 251
description (*атрибут Group*), 218, 255
description (*атрибут GroupsGetById*), 124, 152
DisabledStorage (*класс в vk_maria.dispatcher.fsm.storage.core*), 85, 90
Dispatcher (*класс в vk_maria.dispatcher.dispatcher*), 65, 66
display_order_id (*атрибут Order*), 232, 270
docs_get_messages_upload_server() (*метод Vk*), 36, 57
docs_get_wall_upload_server() (*метод Vk*), 36, 57
docs_save() (*метод Vk*), 36, 57
docs_search() (*метод Vk*), 36, 57
DocsGetMessagesUploadServer (*класс в vk_maria.responses*), 120, 153
DocsGetWallUploadServer (*класс в vk_maria.responses*), 120, 153
DocsSave (*класс в vk_maria.responses*), 120, 154
DocsSearch (*класс в vk_maria.responses*), 121, 153
Document (*класс в vk_maria.vk_types*), 212, 270
document() (*метод Upload*), 190, 191
domain (*атрибут Profile*), 240, 267

E

education (*атрибут Profile*), 240, 267
Education (*класс в vk_maria.vk_types*), 213, 260
education_form (*атрибут Universities*), 248, 266
education_status (*атрибут Universities*), 248, 266
email (*атрибут Contacts*), 202
empty() (*метод MessageInfo*), 183, 185
enabled (*атрибут Cover*), 209, 250
enabled (*атрибут Market*), 225, 253
end_date (*атрибут BanInfo*), 198, 269
error_catcher() (*в модуле vk_maria.utils*), 193
Event (*класс в vk_maria.types.event*), 162
event_handler() (*метод Dispatcher*), 65, 66
event_id (*атрибут CallbackQuery*), 179, 187
event_id (*атрибут CallbackQueryEvent*), 160
EventType (*класс в vk_maria.types.event_type*), 163, 166
EventTypeFilter (*класс в vk_maria.dispatcher.filters.filters*), 70, 76

exceptions (*атрибут VkMariaException*), 114, 116
exports (*атрибут Profile*), 240, 267
ext (*атрибут DocsSave*), 121, 154
ext (*атрибут Document*), 213, 270

F

faculty (*атрибут Education*), 214, 260
faculty (*атрибут Universities*), 248, 265
faculty_name (*атрибут Education*), 214, 260
faculty_name (*атрибут Universities*), 248, 265
FileStorage (*класс в vk_maria.dispatcher.fsm.storage.file.base*), 93, 95
Filters (*класс в vk_maria.dispatcher.filters.filters*), 71, 77
FiltersFactory (*класс в vk_maria.dispatcher.filters.filters*), 72, 77
FiltersFactory.UnknownFilterException, 72, 77
finish() (*метод класса StatesGroup*), 81, 82
finish() (*метод BaseStorage*), 84, 89
finish() (*метод DisabledStorage*), 86, 91
finish() (*метод FileStorage*), 94, 96
finish() (*метод FSMContext*), 88, 90
finish() (*метод JSONStorage*), 99, 100
finish() (*метод MemoryStorage*), 109, 111
finish() (*метод PickleStorage*), 104, 105
finish_date (*атрибут Group*), 218, 256
finish_date (*атрибут GroupsGetById*), 124, 152
first() (*метод класса StatesGroup*), 81, 82
first_name (*атрибут Profile*), 239, 266
first_name_abl (*атрибут Profile*), 240, 267
first_name_acc (*атрибут Profile*), 240, 267
first_name_dat (*атрибут Profile*), 240, 267
first_name_gen (*атрибут Profile*), 240, 267
first_name_ins (*атрибут Profile*), 240, 267
first_name_nom (*атрибут Profile*), 240, 267
fixed_post (*атрибут Group*), 218, 255
fixed_post (*атрибут GroupsGetById*), 124, 152
followers (*атрибут Counters*), 205, 259
followers_count (*атрибут Profile*), 240, 267
friend_status (*атрибут Profile*), 240, 267
friends (*атрибут Counters*), 205, 258
from_ (*атрибут Career*), 199, 256
from_ (*атрибут Military*), 227, 261
from_chat (*атрибут Message*), 180, 185
from_chat (*атрибут MessageEvent*), 188
from_group (*атрибут Message*), 180, 185
from_group (*атрибут MessageEvent*), 188
from_id (*атрибут MessageInfo*), 183, 184
from_id (*атрибут MessagesPin*), 138, 155
from_user (*атрибут Message*), 180, 185
from_user (*атрибут MessageEvent*), 188
fromkeys() (*метод класса MessageInfo*), 184, 185
fromkeys() (*метод Adresses*), 196, 249
fromkeys() (*метод Career*), 199, 256
fromkeys() (*метод City*), 201, 257
fromkeys() (*метод Contacts*), 202, 258
fromkeys() (*метод Counters*), 205, 259
fromkeys() (*метод Country*), 207, 259
fromkeys() (*метод Cover*), 209, 250
fromkeys() (*метод Currency*), 211, 252
fromkeys() (*метод Education*), 214, 260
fromkeys() (*метод Images*), 220, 250
fromkeys() (*метод LastSeen*), 221, 261
fromkeys() (*метод Links*), 224, 251
fromkeys() (*метод Market*), 226, 253
fromkeys() (*метод Military*), 228, 262
fromkeys() (*метод Occupation*), 229, 262
fromkeys() (*метод Personal*), 233, 263
fromkeys() (*метод Place*), 235, 254
fromkeys() (*метод Relatives*), 243, 264
fromkeys() (*метод Schools*), 245, 265
fromkeys() (*метод Universities*), 248, 266
FSMContext (*класс в vk_maria.dispatcher.fsm.storage.core*), 87, 90
FSMStateFilter (*класс в vk_maria.dispatcher.filters.filters*), 71, 77
FunctionFilter (*класс в vk_maria.dispatcher.filters.filters*), 72, 76
fwd_messages (*атрибут MessageInfo*), 183, 184
fwd_messages (*атрибут MessagesPin*), 138, 155

G

games (*атрибут Profile*), 240, 267
geo (*атрибут MessagesPin*), 138, 155
get() (*метод Adresses*), 197, 249
get() (*метод Career*), 199, 256
get() (*метод City*), 201, 257
get() (*метод Contacts*), 203, 258
get() (*метод Counters*), 206, 259
get() (*метод Country*), 207, 259
get() (*метод Cover*), 209, 251
get() (*метод Currency*), 211, 252
get() (*метод Education*), 214, 260
get() (*метод Images*), 220, 250
get() (*метод LastSeen*), 221, 261
get() (*метод Links*), 224, 251
get() (*метод Market*), 226, 253
get() (*метод MessageInfo*), 184, 185
get() (*метод Military*), 228, 262
get() (*метод Occupation*), 229, 262
get() (*метод Personal*), 233, 263
get() (*метод Place*), 235, 254
get() (*метод Relatives*), 243, 264

get() (*метод Schools*), 245, 265
get() (*метод Universities*), 248, 266
get_chat_id() (*метод класса Chat*), 161, 162
get_data() (*метод BaseStorage*), 84, 89
get_data() (*метод DisabledStorage*), 86, 91
get_data() (*метод FileStorage*), 94, 96
get_data() (*метод FSMContext*), 88, 90
get_data() (*метод JSONStorage*), 99, 101
get_data() (*метод MemoryStorage*), 108, 110
get_data() (*метод PickleStorage*), 104, 106
get_filter_by_key() (*метод класса FiltersFactory*), 72, 78
get_filters() (*метод класса FiltersFactory*), 72, 78
get_json() (*метод KeyboardMarkup*), 171, 178
get_random_id() (*в модуле vk_maria.utils*), 193
get_state() (*метод BaseStorage*), 83, 88
get_state() (*метод DisabledStorage*), 85, 90
get_state() (*метод FileStorage*), 94, 96
get_state() (*метод FSMContext*), 87, 90
get_state() (*метод JSONStorage*), 99, 101
get_state() (*метод MemoryStorage*), 108, 110
get_state() (*метод PickleStorage*), 104, 106
get_user_id() (*метод класса Chat*), 161, 162
graduation (*атрибут Education*), 214, 260
graduation (*атрибут Universities*), 248, 266
Group (*класс в vk_maria.vk_types*), 215, 254
GROUP_CHANGE_PHOTO (*атрибут EventType*), 166, 167
GROUP_CHANGE_SETTINGS (*атрибут EventType*), 166, 167
group_id (*атрибут Career*), 199, 256
group_id (*атрибут Order*), 231, 269
group_id (*атрибут PhotosGetMessagesUploadServer*), 140, 156
GROUP_JOIN (*атрибут EventType*), 165, 167
GROUP_LEAVE (*атрибут EventType*), 165, 167
GROUP_OFFICERS_EDIT (*атрибут EventType*), 165, 167
groups (*атрибут Counters*), 205, 258
groups (*атрибут MessagesGetConversationMembers*), 131, 154
groups (*атрибут MessagesGetConversations*), 132, 155
groups (*атрибут MessagesGetLongpollHistory*), 136, 156
groups (*атрибут StoriesGet*), 144, 157
groups (*атрибут StoriesGetById*), 145, 157
groups (*атрибут StoriesGetReplies*), 146, 158
groups_add_address() (*метод Vk*), 31, 52
groups_delete_address() (*метод Vk*), 32, 53
groups_disable_online() (*метод Vk*), 32, 53
groups_edit() (*метод Vk*), 32, 53
groups_edit_address() (*метод Vk*), 34, 55
groups_enable_online() (*метод Vk*), 35, 56
groups_get_banned() (*метод Vk*), 35, 56
groups_get_by_id() (*метод Vk*), 35, 56
groups_get_longpoll_server() (*метод Vk*), 36, 57
groups_get_members() (*метод Vk*), 35, 56
groups_get_online_status() (*метод Vk*), 35, 56
groups_get_token_permissions() (*метод Vk*), 35, 57
groups_is_member() (*метод Vk*), 35, 56
groups_set_settings() (*метод Vk*), 36, 57
GroupsGetBanned (*класс в vk_maria.responses*), 122, 151
GroupsGetById (*класс в vk_maria.responses*), 122, 151
GroupsGetMembers (*класс в vk_maria.responses*), 126, 153
GroupsGetOnlineStatus (*класс в vk_maria.responses*), 126, 153
GroupsGetTokenPermissions (*класс в vk_maria.responses*), 127, 153
GroupsIsMember (*класс в vk_maria.responses*), 127, 153

H

HandlerManager (*класс в vk_maria.dispatcher.filters.handler*), 78, 79
HandlerObject (*класс в vk_maria.dispatcher.filters.handler*), 78, 79
has_key() (*метод MessageInfo*), 184, 185
has_mobile (*атрибут Profile*), 240, 267
has_photo (*атрибут Group*), 218, 255
has_photo (*атрибут GroupsGetById*), 124, 152
has_photo (*атрибут Profile*), 240, 268
height (*атрибут Images*), 219, 250
history (*атрибут MessagesGetLongpollHistory*), 136, 156
home_phone (*атрибут Contacts*), 202, 258
home_town (*атрибут Profile*), 240, 268
http (*атрибут ApiMethod*), 15, 43

I

id (*атрибут City*), 201, 257
id (*атрибут Country*), 207, 259
id (*атрибут Currency*), 211, 252
id (*атрибут DocsSave*), 121, 154
id (*атрибут Document*), 212, 270
id (*атрибут Group*), 217, 254
id (*атрибут GroupsGetById*), 125, 152
id (*атрибут Links*), 223, 251
id (*атрибут MessageInfo*), 183, 184
id (*атрибут MessagesPin*), 138, 155
id (*атрибут Occupation*), 229, 262
id (*атрибут Order*), 231, 269

id (*ампибут PhotosSaveMessagesPhoto*), 142, 157
id (*ампибут Place*), 235, 253
id (*ампибут Profile*), 239, 266
id (*ампибут Relatives*), 243, 264
id (*ампибут Schools*), 245, 264
id (*ампибут Universities*), 248, 265
images (*ампибут Cover*), 209, 250
images (*ампибут PhotosSaveOwnerCoverPhoto*), 143, 157
Images (*класс в vk_maria.vk_types*), 219, 250
important (*ампибут MessageInfo*), 183, 184
in_read (*ампибут MessagesGetHistory*), 133, 155
inline (*ампибут KeyboardModel*), 172, 177
inspired_by (*ампибут Personal*), 233, 263
interests (*ампибут Profile*), 240, 268
InvalidFileFormatError, 189
InvalidParametersError, 113, 115
invitation (*ампибут GroupsIsMember*), 128, 153
invited_by (*ампибут Group*), 217, 255
invited_by (*ампибут GroupsGetById*), 125, 153
is_admin (*ампибут Group*), 217, 254
is_admin (*ампибут GroupsGetById*), 125, 153
is_advertiser (*ампибут Group*), 217, 255
is_advertiser (*ампибут GroupsGetById*), 125, 153
is_allowed (*ампибут MessagesIsMessagesFromGroupAllowed*), 137, 155
is_closed (*ампибут Group*), 217, 254
is_closed (*ампибут GroupsGetById*), 125, 152
is_closed (*ампибут Profile*), 239, 266
is_enabled (*ампибут Adresses*), 196, 249
is_favorite (*ампибут Group*), 218, 255
is_favorite (*ампибут GroupsGetById*), 124, 152
is_favorite (*ампибут Profile*), 240, 268
is_friend (*ампибут Profile*), 241, 268
is_hidden (*ампибут MessageInfo*), 183, 185
is_hidden_from_feed (*ампибут Group*), 218, 255
is_hidden_from_feed (*ампибут GroupsGetById*), 124, 152
is_hidden_from_feed (*ампибут Profile*), 241, 268
is_main_variant (*ампибут Order*), 231, 269
is_member (*ампибут Group*), 217, 254
is_member (*ампибут GroupsGetById*), 125, 153
is_messages_blocked (*ампибут Group*), 218, 255
is_messages_blocked (*ампибут GroupsGetById*), 124, 152
items (*ампибут DocsSearch*), 121, 154
items (*ампибут GroupsGetBanned*), 122, 151
items (*ампибут GroupsGetMembers*), 126, 153
items (*ампибут MarketGetGroupOrders*), 128, 156
items (*ампибут MarketGetOrderItems*), 129, 156
items (*ампибут MessagesGetByConversationMessageId*), 130, 154
items (*ампибут MessagesGetById*), 131, 154
items (*ампибут MessagesGetConversationMembers*), 131, 154
items (*ампибут MessagesGetConversations*), 132, 154
items (*ампибут MessagesGetConversationsById*), 133, 155
items (*ампибут MessagesGetHistory*), 133, 155
items (*ампибут MessagesGetHistoryAttachments*), 134, 155
items (*ампибут MessagesGetIntentUsers*), 135, 155
items (*ампибут MessagesSearch*), 138, 156
items (*ампибут MessagesSearchConversations*), 139, 156
items (*ампибут StoriesGet*), 144, 157
items (*ампибут StoriesGetById*), 145, 157
items (*ампибут StoriesGetReplies*), 146, 157
items (*ампибут StoriesGetViewers*), 148, 158
items (*ампибут StoriesSave*), 148, 158
items() (*метод Adresses*), 197, 249
items() (*метод Career*), 199, 256
items() (*метод City*), 201, 257
items() (*метод Contacts*), 203, 258
items() (*метод Counters*), 206, 259
items() (*метод Country*), 207, 259
items() (*метод Cover*), 209, 251
items() (*метод Currency*), 211, 252
items() (*метод Education*), 214, 260
items() (*метод Images*), 220, 250
items() (*метод LastSeen*), 221, 261
items() (*метод Links*), 224, 251
items() (*метод Market*), 226, 253
items() (*метод MessageInfo*), 184, 185
items() (*метод Military*), 228, 262
items() (*метод Occupation*), 230, 262
items() (*метод Personal*), 233, 263
items() (*метод Place*), 235, 254
items() (*метод Relatives*), 243, 264
items() (*метод Schools*), 245, 265
items() (*метод Universities*), 248, 266
items_count (*ампибут Order*), 231, 269

J

JSONStorage (*класс в vk_maria.dispatcher.fsm.storage.file.json*), 97, 100

K

key (*ампибут BoundFilter*), 68, 75
key (*ампибут CommandsFilter*), 70, 76
key (*ампибут EventTypeFilter*), 70, 76
key (*ампибут FSMStateFilter*), 71, 77
key (*ампибут FunctionFilter*), 72, 76

key (*атрибут MessagesGetLongpollServer*), 136, 156
key (*атрибут PayloadFilter*), 73, 77
key (*атрибут RegexpFilter*), 74, 77
key (*атрибут TextFilter*), 74, 76
key (*атрибут TypeFromFilter*), 75, 77
key (*атрибут UtilsGetLinkStats*), 149, 158
key (*атрибут UtilsGetShortLink*), 150, 158
KeyboardMarkup (*класс в vk_maria.types.keyboard*), 171, 177
KeyboardModel (*класс в vk_maria.types.keyboard*), 171, 177
KeyboardModelMeta (*класс в vk_maria.types.keyboard*), 172, 177
KeyIsNotValidError, 113, 116
keys() (*метод Adresses*), 197, 249
keys() (*метод Career*), 199, 256
keys() (*метод City*), 201, 257
keys() (*метод Contacts*), 203, 258
keys() (*метод Counters*), 206, 259
keys() (*метод Country*), 207, 259
keys() (*метод Cover*), 209, 251
keys() (*метод Currency*), 211, 252
keys() (*метод Education*), 214, 260
keys() (*метод Images*), 220, 250
keys() (*метод LastSeen*), 221, 261
keys() (*метод Links*), 224, 251
keys() (*метод Market*), 226, 253
keys() (*метод MessageInfo*), 184, 185
keys() (*метод Military*), 228, 262
keys() (*метод Occupation*), 230, 262
keys() (*метод Personal*), 233, 263
keys() (*метод Place*), 235, 254
keys() (*метод Relatives*), 243, 264
keys() (*метод Schools*), 246, 265
keys() (*метод Universities*), 249, 266

L

langs (*атрибут Personal*), 233, 263
last() (*метод класса StatesGroup*), 81, 82
last_deleted_id (*атрибут MessagesDeleteConversation*), 130, 154
last_name (*атрибут Profile*), 239, 266
last_name_abl (*атрибут Profile*), 241, 268
last_name_acc (*атрибут Profile*), 241, 268
last_name_dat (*атрибут Profile*), 241, 268
last_name_gen (*атрибут Profile*), 241, 268
last_name_ins (*атрибут Profile*), 241, 268
last_name_nom (*атрибут Profile*), 241, 268
last_request (*атрибут ApiMethod*), 15, 43
last_seen (*атрибут Profile*), 241, 268
LastSeen (*класс в vk_maria.vk_types*), 220, 261
latitude (*атрибут Place*), 235, 253
life_main (*атрибут Personal*), 233, 263
link (*атрибут MessagesGetInviteLink*), 135, 155
link (*атрибут UtilsCheckLink*), 149, 158
links (*атрибут Group*), 218, 255
links (*атрибут GroupsGetById*), 124, 152
Links (*класс в vk_maria.vk_types*), 222, 251
listen() (*метод LongPoll*), 117
lists (*атрибут Profile*), 241, 268
Location (*атрибут Button*), 169, 177
LocationButton (*класс в vk_maria.types.keyboard*), 173, 176
longitude (*атрибут Place*), 235, 253
LongPoll (*класс в vk_maria.longpoll.longpoll*), 117

M

maiden_name (*атрибут Profile*), 241, 268
main_address_id (*атрибут Adresses*), 196, 249
main_album_id (*атрибут Group*), 218, 255
main_album_id (*атрибут GroupsGetById*), 125, 152
main_album_id (*атрибут Market*), 226, 253
main_section (*атрибут Group*), 218, 255
main_section (*атрибут GroupsGetById*), 125, 152
market (*атрибут Group*), 218, 256
market (*атрибут GroupsGetById*), 125, 152
Market (*класс в vk_maria.vk_types*), 224, 252
MARKET_COMMENT_DELETE (*атрибут EventType*), 165, 167
MARKET_COMMENT_EDIT (*атрибут EventType*), 165, 167
MARKET_COMMENT_NEW (*атрибут EventType*), 165, 167
MARKET_COMMENT_RESTORE (*атрибут EventType*), 165, 167
market_edit_order() (*метод Vk*), 37, 58
market_get_group_orders() (*метод Vk*), 37, 58
market_get_order_by_id() (*метод Vk*), 37, 58
market_get_order_items() (*метод Vk*), 38, 59
MarketGetGroupOrders (*класс в vk_maria.responses*), 128, 156
MarketGetOrderItems (*класс в vk_maria.responses*), 128, 156
mask (*атрибут GroupsGetTokenPermissions*), 127, 153
member (*атрибут GroupsIsMember*), 127, 153
member_status (*атрибут Group*), 218, 256
member_status (*атрибут GroupsGetById*), 125, 152
members_count (*атрибут Group*), 218, 256
members_count (*атрибут GroupsGetById*), 125, 152
MemoryStorage (*класс в vk_maria.dispatcher.fsm.storage.memory.memory*), 107, 110
message (*атрибут Message*), 180, 185

message (*атрибут MessageEvent*), 188
Message (*класс в vk_maria.types.message*), 180, 185
MESSAGE_ALLOW (*атрибут EventType*), 164, 166
MESSAGE_DENY (*атрибут EventType*), 164, 166
MESSAGE_EDIT (*атрибут EventType*), 164, 166
MESSAGE_EVENT (*атрибут EventType*), 164, 166
message_handler() (*метод Dispatcher*), 66
message_id (*атрибут MessagesDeleteChatPhoto*), 129, 154
message_id (*атрибут MessagesSetChatPhoto*), 139, 156
MESSAGE_NEW (*атрибут EventType*), 164, 166
MESSAGE_REPLY (*атрибут EventType*), 164, 166
MESSAGE_TYPING_STATE (*атрибут EventType*), 164, 166
MessageEvent (*класс в vk_maria.types.message_event*), 187, 188
MessageInfo (*класс в vk_maria.types.message*), 182, 184
messages (*атрибут MessagesGetImportantMessages*), 134, 155
messages (*атрибут MessagesGetLongpollHistory*), 136, 156
messages_create_chat() (*метод Vk*), 22, 43
messages_delete() (*метод Vk*), 22, 43
messages_delete_chat_photo() (*метод Vk*), 23, 44
messages_delete_conversation() (*метод Vk*), 23, 44
messages_edit() (*метод Vk*), 23, 44
messages_edit_chat() (*метод Vk*), 23, 45
messages_get_by_conversation_message_id() (*метод Vk*), 24, 45
messages_get_by_id() (*метод Vk*), 24, 45
messages_get_conversation_members() (*метод Vk*), 24, 45
messages_get_conversations() (*метод Vk*), 24, 45
messages_get_conversations_by_id() (*метод Vk*), 25, 46
messages_get_history() (*метод Vk*), 25, 46
messages_get_history_attachments() (*метод Vk*), 25, 46
messages_get_important_messages() (*метод Vk*), 26, 47
messages_get_intent_users() (*метод Vk*), 26, 47
messages_get_invite_link() (*метод Vk*), 27, 48
messages_get_longpoll_history() (*метод Vk*), 27, 48
messages_get_longpoll_server() (*метод Vk*), 28, 49
messages_is_messages_from_group_allowed() (*метод Vk*), 28, 49
messages_mark_as_answered_conversation() (*метод Vk*), 28, 49
messages_mark_as_important_conversation() (*метод Vk*), 28, 49
messages_mark_as_read() (*метод Vk*), 28, 49
messages_pin() (*метод Vk*), 29, 50
messages_remove_chat_user() (*метод Vk*), 29, 50
messages_restore() (*метод Vk*), 29, 50
messages_search() (*метод Vk*), 29, 50
messages_search_conversations() (*метод Vk*), 29, 51
messages_send() (*метод Vk*), 30, 51
messages_send_message_event_answer() (*метод Vk*), 31, 52
messages_set_activity() (*метод Vk*), 31, 52
messages_set_chat_photo() (*метод Vk*), 31, 52
messages_unpin() (*метод Vk*), 31, 52
MessagesDeleteChatPhoto (*класс в vk_maria.responses*), 129, 154
MessagesDeleteConversation (*класс в vk_maria.responses*), 129, 154
MessagesGetByConversationMessageId (*класс в vk_maria.responses*), 130, 154
MessagesGetById (*класс в vk_maria.responses*), 130, 154
MessagesGetConversationMembers (*класс в vk_maria.responses*), 131, 154
MessagesGetConversations (*класс в vk_maria.responses*), 132, 154
MessagesGetConversationsById (*класс в vk_maria.responses*), 132, 155
MessagesGetHistory (*класс в vk_maria.responses*), 133, 155
MessagesGetHistoryAttachments (*класс в vk_maria.responses*), 134, 155
MessagesGetImportantMessages (*класс в vk_maria.responses*), 134, 155
MessagesGetIntentUsers (*класс в vk_maria.responses*), 135, 155
MessagesGetInviteLink (*класс в vk_maria.responses*), 135, 155
MessagesGetLongpollHistory (*класс в vk_maria.responses*), 136, 156
MessagesGetLongpollServer (*класс в vk_maria.responses*), 136, 156
MessagesIsMessagesFromGroupAllowed (*класс в vk_maria.responses*), 137, 155
MessagesPin (*класс в vk_maria.responses*), 137, 155
MessagesSearch (*класс в vk_maria.responses*), 138, 155
MessagesSearchConversations (*класс в vk_maria.responses*), 139, 156
MessagesSetChatPhoto (*класс в vk_maria.responses*), 139, 156

military (*атрибут Profile*), 241, 268
Military (*класс в vk_maria.vk_types*), 226, 261
minutes (*атрибут GroupsGetOnlineStatus*), 126, 153
mobile_phone (*атрибут Contacts*), 202, 258
move_to_end() (*метод MessageInfo*), 184, 185
movies (*атрибут Profile*), 241, 268
mro() (*метод BoundFilterMeta*), 69, 75
mro() (*метод KeyboardModelMeta*), 172, 177
mro() (*метод StatesGroupMeta*), 81
music (*атрибут Profile*), 241, 268
mutual_friends (*атрибут Counters*), 205, 258

N

name (*атрибут Currency*), 211, 252
name (*атрибут Group*), 217, 254
name (*атрибут GroupsGetById*), 125, 152
name (*атрибут Links*), 223, 251
name (*атрибут Occupation*), 229, 262
name (*атрибут Relatives*), 243, 264
name (*атрибут Schools*), 245, 264
name (*атрибут Universities*), 248, 265
NEGATIVE (*атрибут Color*), 170, 177
next() (*метод класса StatesGroup*), 81, 82
next_from (*атрибут MessagesGetHistoryAttachments*), 134, 155
nickname (*атрибут Profile*), 241, 268
notes (*атрибут Counters*), 205, 258

O

object_id (*атрибут UtilsResolveScreenName*), 150, 159
occupation (*атрибут Profile*), 241, 268
Occupation (*класс в vk_maria.vk_types*), 228, 262
one_time (*атрибут KeyboardModel*), 172, 177
online (*атрибут Profile*), 241, 268
online_friends (*атрибут Counters*), 205, 258
open_file() (*в модуле vk_maria.upload.utils*), 192
open_files() (*в модуле vk_maria.upload.utils*), 192
open_link (*атрибут StoriesGetStats*), 147, 158
OpenLink (*атрибут Button*), 169, 177
OpenLinkButton (*класс в vk_maria.types.keyboard*), 173, 175
Order (*класс в vk_maria.vk_types*), 230, 269
out (*атрибут MessageInfo*), 183, 184
out_read (*атрибут MessagesGetHistory*), 133, 155
owner_id (*атрибут DocsSave*), 121, 154
owner_id (*атрибут Document*), 213, 270
owner_id (*атрибут PhotosSaveMessagesPhoto*), 142, 157

P

pages (*атрибут Counters*), 205, 259
parent_stack (*атрибут WallCreateComment*), 151, 159
payload (*атрибут CallbackQuery*), 179, 187
payload (*атрибут CallbackQueryEvent*), 160
PayloadFilter (*класс в vk_maria.dispatcher.filters.filters*), 73, 77
peer_id (*атрибут CallbackQuery*), 179, 187
peer_id (*атрибут CallbackQueryEvent*), 160
peer_id (*атрибут Message*), 180, 185
peer_id (*атрибут MessageEvent*), 188
peer_id (*атрибут MessageInfo*), 183, 184
people_main (*атрибут Personal*), 233, 263
PermissionError, 113, 115
personal (*атрибут Profile*), 241, 268
Personal (*класс в vk_maria.vk_types*), 232, 263
phone (*атрибут Contacts*), 202
photo() (*метод Upload*), 190, 191
photo_100 (*атрибут Group*), 217, 255
photo_100 (*атрибут GroupsGetById*), 125, 153
photo_100 (*атрибут Links*), 224, 251
photo_100 (*атрибут Profile*), 241, 268
photo_200 (*атрибут Group*), 217, 255
photo_200 (*атрибут GroupsGetById*), 125, 153
photo_200 (*атрибут Profile*), 241, 268
photo_200_orig (*атрибут Profile*), 241, 268
photo_400_orig (*атрибут Profile*), 241, 268
photo_50 (*атрибут Group*), 217, 255
photo_50 (*атрибут GroupsGetById*), 125, 153
photo_50 (*атрибут Links*), 223, 251
photo_50 (*атрибут Profile*), 241, 268
PHOTO_COMMENT_DELETE (*атрибут EventType*), 165, 166
PHOTO_COMMENT_EDIT (*атрибут EventType*), 165, 166
PHOTO_COMMENT_NEW (*атрибут EventType*), 165, 166
PHOTO_COMMENT_RESTORE (*атрибут EventType*), 165, 166
photo_id (*атрибут Profile*), 241, 268
photo_max (*атрибут Profile*), 241, 268
photo_max_orig (*атрибут Profile*), 241, 268
PHOTO_NEW (*атрибут EventType*), 165, 166
photos (*атрибут Counters*), 205, 258
photos_get_chat_upload_server() (*метод Vk*), 38, 59
photos_get_messages_upload_server() (*метод Vk*), 38, 59
photos_get_owner_cover_photo_upload_server() (*метод Vk*), 38, 59
photos_save_messages_photo() (*метод Vk*), 38, 59

photos_save_owner_cover_photo() (*метод Vk*), 39, 60
PhotosGetChatUploadServer (*класс в vk_maria.responses*), 140, 156
PhotosGetMessagesUploadServer (*класс в vk_maria.responses*), 140, 156
PhotosGetOwnerCoverPhotoUploadServer (*класс в vk_maria.responses*), 141, 156
PhotosSaveMessagesPhoto (*класс в vk_maria.responses*), 141, 157
PhotosSaveOwnerCoverPhoto (*класс в vk_maria.responses*), 142, 157
PickleStorage (*класс в vk_maria.dispatcher.fsm.storage.file.pickle*), 102, 105
pid (*атрибут PhotosSaveMessagesPhoto*), 142, 157
place (*атрибут Group*), 218, 256
place (*атрибут GroupsGetById*), 125, 152
Place (*класс в vk_maria.vk_types*), 234, 253
platform (*атрибут LastSeen*), 221, 261
podcasts (*атрибут PodcastsSearchPodcast*), 143, 157
podcasts_search_podcast() (*метод Vk*), 39, 60
PodcastsSearchPodcast (*класс в vk_maria.responses*), 143, 157
political (*атрибут Personal*), 233, 263
POLL_VOTE_NEW (*атрибут EventType*), 165, 167
pop() (*метод Adresses*), 197, 249
pop() (*метод Career*), 199, 256
pop() (*метод City*), 201, 257
pop() (*метод Contacts*), 203, 258
pop() (*метод Counters*), 206, 259
pop() (*метод Country*), 208, 259
pop() (*метод Cover*), 210, 251
pop() (*метод Currency*), 212, 252
pop() (*метод Education*), 214, 260
pop() (*метод Images*), 220, 250
pop() (*метод LastSeen*), 222, 261
pop() (*метод Links*), 224, 251
pop() (*метод Market*), 226, 253
pop() (*метод MessageInfo*), 184, 185
pop() (*метод Military*), 228, 262
pop() (*метод Occupation*), 230, 262
pop() (*метод Personal*), 233, 263
pop() (*метод Place*), 235, 254
pop() (*метод Relatives*), 243, 264
pop() (*метод Schools*), 246, 265
pop() (*метод Universities*), 249, 266
popitem() (*метод Adresses*), 197, 249
popitem() (*метод Career*), 199, 257
popitem() (*метод City*), 201, 257
popitem() (*метод Contacts*), 203, 258
popitem() (*метод Counters*), 206, 259
popitem() (*метод Country*), 208, 260
popitem() (*метод Cover*), 210, 251
popitem() (*метод Currency*), 212, 252
popitem() (*метод Education*), 214, 260
popitem() (*метод Images*), 220, 250
popitem() (*метод LastSeen*), 222, 261
popitem() (*метод Links*), 224, 252
popitem() (*метод Market*), 226, 253
popitem() (*метод MessageInfo*), 184, 185
popitem() (*метод Military*), 228, 262
popitem() (*метод Occupation*), 230, 262
popitem() (*метод Personal*), 233, 263
popitem() (*метод Place*), 235, 254
popitem() (*метод Relatives*), 243, 264
popitem() (*метод Schools*), 246, 265
popitem() (*метод Universities*), 249, 266
position (*атрибут Career*), 199, 256
POSITIVE (*атрибут Color*), 170, 177
preview (*атрибут DocsSave*), 121, 154
preview (*атрибут Document*), 213, 270
preview_order_items (*атрибут Order*), 232, 270
previous() (*метод класса StatesGroup*), 81, 82
price_max (*атрибут Market*), 226, 253
price_min (*атрибут Market*), 225, 253
PRIMARY (*атрибут Color*), 170, 177
Profile (*класс в vk_maria.vk_types*), 236, 266
profiles (*атрибут MessagesGetConversationMembers*), 131, 154
profiles (*атрибут MessagesGetConversations*), 132, 155
profiles (*атрибут MessagesGetLongpollHistory*), 136, 156
profiles (*атрибут StoriesGet*), 144, 157
profiles (*атрибут StoriesGetById*), 145, 157
profiles (*атрибут StoriesGetReplies*), 146, 158
property_values (*атрибут Order*), 231, 269
public_date_label (*атрибут Group*), 218, 256
public_date_label (*атрибут GroupsGetById*), 125, 152

Q

query_delimiter() (*в модуле vk_maria.utils*), 193
quotes (*атрибут Profile*), 241, 268

R

random_id (*атрибут MessageInfo*), 183, 184
read() (*метод FileStorage*), 93, 95
read() (*метод JSONStorage*), 98, 100
read() (*метод PickleStorage*), 103, 105
reason (*атрибут BanInfo*), 198, 269
recipient (*атрибут Order*), 232, 270
RegexpFilter (*класс в vk_maria.dispatcher.filters.filters*), 73, 77

register_callback_handler() (*метод Dispatcher*), 65, 66
register_event_handler() (*метод Dispatcher*), 65, 66
register_handler() (*метод HandlerManager*), 78, 79
register_message_handler() (*метод Dispatcher*), 65, 66
registered_filters (*атрибут BoundFilterMeta*), 69, 75
relation (*атрибут Profile*), 241, 269
relatives (*атрибут Profile*), 241, 269
Relatives (*класс в vk_maria.vk_types*), 242, 264
religion (*атрибут Personal*), 233, 263
replies (*атрибут StoriesGetStats*), 147, 158
reply() (*метод Message*), 181, 186
reply() (*метод MessageEvent*), 188
request (*атрибут GroupsIsMember*), 127, 153
reset_data() (*метод BaseStorage*), 84, 89
reset_data() (*метод DisabledStorage*), 87, 92
reset_data() (*метод FileStorage*), 94, 96
reset_data() (*метод FSMContext*), 88, 90
reset_data() (*метод JSONStorage*), 99, 101
reset_data() (*метод MemoryStorage*), 109, 111
reset_data() (*метод PickleStorage*), 104, 106
reset_state() (*метод BaseStorage*), 84, 89
reset_state() (*метод DisabledStorage*), 87, 92
reset_state() (*метод FileStorage*), 94, 96
reset_state() (*метод FSMContext*), 88, 90
reset_state() (*метод JSONStorage*), 99, 101
reset_state() (*метод MemoryStorage*), 109, 111
reset_state() (*метод PickleStorage*), 104, 106
resolve_address() (*метод FileStorage*), 95, 96
resolve_address() (*метод JSONStorage*), 99, 101
resolve_address() (*метод MemoryStorage*), 108, 110
resolve_address() (*метод PickleStorage*), 104, 106
resolve_address() (*статический метод Chat*), 161, 162
Response (*класс в vk_maria.responses*), 143, 151
response_parser() (*в модуле vk_maria.utils*), 193
ResponseItem (*класс в vk_maria.responses*), 144, 151
results_total (*атрибут PodcastsSearchPodcast*), 143, 157
row1 (*атрибут KeyboardModel*), 172, 177
row2 (*атрибут KeyboardModel*), 172, 177
row3 (*атрибут KeyboardModel*), 172, 177
row4 (*атрибут KeyboardModel*), 172, 177
row5 (*атрибут KeyboardModel*), 172, 177
rps_delay (*атрибут ApiMethod*), 15, 43

S

schools (*атрибут Profile*), 241, 269
Schools (*класс в vk_maria.vk_types*), 244, 264
screen_name (*атрибут Group*), 217, 254
screen_name (*атрибут GroupsGetById*), 125, 152
screen_name (*атрибут Profile*), 241, 269
SECONDARY (*атрибут Color*), 170, 177
server (*атрибут MessagesGetLongpollServer*), 137, 156
ServerError, 113, 115
set() (*метод класса Chat*), 161, 162
set() (*метод State*), 80, 82
set_chat_photo() (*метод Upload*), 190, 191
set_data() (*метод BaseStorage*), 84, 89
set_data() (*метод DisabledStorage*), 86, 91
set_data() (*метод FileStorage*), 95, 96
set_data() (*метод FSMContext*), 88, 90
set_data() (*метод JSONStorage*), 99, 101
set_data() (*метод MemoryStorage*), 109, 110
set_data() (*метод PickleStorage*), 104, 106
set_group_cover_photo() (*метод Upload*), 190, 191
set_state() (*метод BaseStorage*), 84, 89
set_state() (*метод DisabledStorage*), 86, 91
set_state() (*метод FileStorage*), 95, 96
set_state() (*метод FSMContext*), 88, 90
set_state() (*метод JSONStorage*), 100, 101
set_state() (*метод MemoryStorage*), 109, 110
set_state() (*метод PickleStorage*), 105, 106
setdefault() (*метод Adresses*), 197, 249
setdefault() (*метод Career*), 199, 257
setdefault() (*метод City*), 201, 257
setdefault() (*метод Contacts*), 203, 258
setdefault() (*метод Counters*), 206, 259
setdefault() (*метод Country*), 208, 260
setdefault() (*метод Cover*), 210, 251
setdefault() (*метод Currency*), 212, 252
setdefault() (*метод Education*), 214, 260
setdefault() (*метод Images*), 220, 250
setdefault() (*метод LastSeen*), 222, 261
setdefault() (*метод Links*), 224, 252
setdefault() (*метод Market*), 226, 253
setdefault() (*метод MessageInfo*), 184, 185
setdefault() (*метод Military*), 228, 262
setdefault() (*метод Occupation*), 230, 263
setdefault() (*метод Personal*), 233, 263
setdefault() (*метод Place*), 235, 254
setdefault() (*метод Relatives*), 243, 264
setdefault() (*метод Schools*), 246, 265
setdefault() (*метод Universities*), 249, 266
settings (*атрибут GroupsGetTokenPermissions*), 127, 153
sex (*атрибут Profile*), 242, 269
shares (*атрибут StoriesGetStats*), 147, 158

short_url (*атрибут UtilsGetShortLink*), 150, 158
Singleton (*класс в vk_maria.mixins*), 117
site (*атрибут Group*), 218, 256
site (*атрибут GroupsGetById*), 125, 152
site (*атрибут Profile*), 242, 269
size (*атрибут DocsSave*), 121, 154
size (*атрибут Document*), 213, 270
smoking (*атрибут Personal*), 233, 263
speciality (*атрибут Schools*), 245, 265
src (*атрибут PhotosSaveMessagesPhoto*), 142, 157
src_big (*атрибут PhotosSaveMessagesPhoto*), 142, 157
src_small (*атрибут PhotosSaveMessagesPhoto*), 142, 157
src_xbig (*атрибут PhotosSaveMessagesPhoto*), 142, 157
src_xxbig (*атрибут PhotosSaveMessagesPhoto*), 142, 157
start_date (*атрибут Group*), 218, 256
start_date (*атрибут GroupsGetById*), 125, 152
start_polling() (*метод Dispatcher*), 66, 67
State (*класс в vk_maria.dispatcher.fsm.state*), 80, 82
state (*State property*), 80, 82
StatesGroup (*класс в vk_maria.dispatcher.fsm.state*), 80, 82
StatesGroupMeta (*класс в vk_maria.dispatcher.fsm.state*), 81
stats (*атрибут UtilsGetLinkStats*), 149, 158
status (*атрибут Group*), 218, 256
status (*атрибут GroupsGetById*), 125, 152
status (*атрибут GroupsGetOnlineStatus*), 126, 153
status (*атрибут Order*), 231, 269
status (*атрибут Profile*), 242, 269
status (*атрибут UtilsCheckLink*), 149, 158
storage_get() (*метод Vk*), 39, 60
storage_get_keys() (*метод Vk*), 39, 60
storage_set() (*метод Vk*), 39, 60
stories_delete() (*метод Vk*), 40, 61
stories_get() (*метод Vk*), 40, 61
stories_get_by_id() (*метод Vk*), 40, 61
stories_get_photo_upload_server() (*метод Vk*), 40, 61
stories_get_replies() (*метод Vk*), 40, 61
stories_get_stats() (*метод Vk*), 41, 62
stories_get_video_upload_server() (*метод Vk*), 41, 62
stories_get_viewers() (*метод Vk*), 41, 62
stories_hide_all_replies() (*метод Vk*), 41, 62
stories_hide_reply() (*метод Vk*), 42, 63
stories_save() (*метод Vk*), 42, 63
StoriesGet (*класс в vk_maria.responses*), 144, 157
StoriesGetById (*класс в vk_maria.responses*), 145, 157
StoriesGetPhotoUploadServer (*класс в vk_maria.responses*), 145, 157
StoriesGetReplies (*класс в vk_maria.responses*), 146, 157
StoriesGetStats (*класс в vk_maria.responses*), 146, 158
StoriesGetVideoUploadServer (*класс в vk_maria.responses*), 147, 158
StoriesGetViewers (*класс в vk_maria.responses*), 148, 158
StoriesSave (*класс в vk_maria.responses*), 148, 158
subscribers (*атрибут StoriesGetStats*), 147, 158

T

test_handler() (*метод HandlerObject*), 79
Text (*атрибут Button*), 169, 177
text (*атрибут MessageInfo*), 183, 184
text (*атрибут MessagesPin*), 138, 155
TextButton (*класс в vk_maria.types.keyboard*), 174, 175
TextFilter (*класс в vk_maria.dispatcher.filters.filters*), 74, 76
time (*атрибут LastSeen*), 221, 261
timezone (*атрибут Profile*), 242, 269
title (*атрибут City*), 201, 257
title (*атрибут Country*), 207, 259
title (*атрибут DocsSave*), 121, 154
title (*атрибут Document*), 213, 270
title (*атрибут Place*), 235, 253
to_dict() (*метод MessageInfo*), 184, 185
total_price (*атрибут Order*), 231, 269
trending (*атрибут Group*), 218, 256
trending (*атрибут GroupsGetById*), 125, 152
trending (*атрибут Profile*), 242, 269
ts (*атрибут MessagesGetLongpollServer*), 137, 156
tv (*атрибут Profile*), 242, 269
type (*атрибут BaseButton*), 169, 175
type (*атрибут BaseEvent*), 178, 184
type (*атрибут CallbackButton*), 170, 176
type (*атрибут CallbackQuery*), 179, 187
type (*атрибут CallbackQueryEvent*), 160
type (*атрибут DocsSave*), 121, 154
type (*атрибут Document*), 213, 270
type (*атрибут Group*), 217, 255
type (*атрибут GroupsGetById*), 125, 153
type (*атрибут LocationButton*), 173, 176
type (*атрибут Market*), 225, 253
type (*атрибут Message*), 181, 186
type (*атрибут MessageEvent*), 188
type (*атрибут Occupation*), 229, 262
type (*атрибут OpenLinkButton*), 173, 176
type (*атрибут Place*), 235, 254
type (*атрибут Relatives*), 243, 264

type (*атрибут Schools*), 245, 265
type (*атрибут TextButton*), 174, 175
type (*атрибут UtilsResolveScreenName*), 150, 158
type (*атрибут VKAppsButton*), 175, 176
type (*атрибут VKPayButton*), 175, 176
type_str (*атрибут Schools*), 245, 265
TypeFromFilter (*класс в vk_maria.dispatcher.filters.filters*), 75, 76

U

unit (*атрибут Military*), 227, 261
unit_id (*атрибут Military*), 227, 261
universities (*атрибут Profile*), 242, 269
Universities (*класс в vk_maria.vk_types*), 246, 265
university (*атрибут Education*), 214, 260
university_name (*атрибут Education*), 214, 260
UnknownError, 114
UnknownMethodError, 114, 115
unpack_button() (*в модуле vk_maria.types.keyboard*), 168, 177
unread_count (*атрибут MessagesGetConversations*), 132, 154
until (*атрибут Career*), 199, 256
until (*атрибут Military*), 227, 261
update() (*метод Adresses*), 197, 249
update() (*метод Career*), 200, 257
update() (*метод City*), 201, 257
update() (*метод Contacts*), 203, 258
update() (*метод Counters*), 206, 259
update() (*метод Country*), 208, 260
update() (*метод Cover*), 210, 251
update() (*метод Currency*), 212, 252
update() (*метод Education*), 214, 260
update() (*метод Images*), 220, 250
update() (*метод LastSeen*), 222, 261
update() (*метод Links*), 224, 252
update() (*метод Market*), 226, 253
update() (*метод MessageInfo*), 184, 185
update() (*метод Military*), 228, 262
update() (*метод Occupation*), 230, 263
update() (*метод Personal*), 234, 263
update() (*метод Place*), 236, 254
update() (*метод Relatives*), 243, 264
update() (*метод Schools*), 246, 265
update() (*метод Universities*), 249, 266
update_data() (*метод BaseStorage*), 84, 89
update_data() (*метод DisabledStorage*), 86, 91
update_data() (*метод FileStorage*), 95, 97
update_data() (*метод FSMContext*), 88, 90
update_data() (*метод JSONStorage*), 100, 101
update_data() (*метод MemoryStorage*), 109, 110
update_data() (*метод PickleStorage*), 105, 106
Upload (*класс в vk_maria.upload.upload*), 190, 191
upload_result (*атрибут StoriesGetPhotoUploadServer*), 145, 157
upload_result (*атрибут StoriesGetVideoUploadServer*), 147, 158
upload_url (*атрибут DocsGetMessagesUploadServer*), 120, 153
upload_url (*атрибут DocsGetWallUploadServer*), 120, 153
upload_url (*атрибут PhotosGetChatUploadServer*), 140, 156
upload_url (*атрибут PhotosGetMessagesUploadServer*), 140, 156
upload_url (*атрибут PhotosGetOwnerCoverPhotoUploadServer*), 141, 156
url (*атрибут DocsSave*), 121, 154
url (*атрибут Document*), 213, 270
url (*атрибут Images*), 219, 250
url (*атрибут Links*), 223, 251
url (*атрибут UtilsGetShortLink*), 150, 158
USER_BLOCK (*атрибут EventType*), 165, 167
user_id (*атрибут CallbackQuery*), 179, 187
user_id (*атрибут CallbackQueryEvent*), 160
user_id (*атрибут Chat*), 161
user_id (*атрибут Contacts*), 202
user_id (*атрибут GroupsIsMember*), 128, 153
user_id (*атрибут Order*), 231, 269
USER_UNBLOCK (*атрибут EventType*), 165, 167
user_videos (*атрибут Counters*), 205, 258
users_get() (*метод Vk*), 39, 60
utils_check_link() (*метод Vk*), 42, 63
utils_get_link_stats() (*метод Vk*), 42, 63
utils_get_server_time() (*метод Vk*), 42, 63
utils_get_short_link() (*метод Vk*), 42, 63
utils_resolve_screen_name() (*метод Vk*), 42, 63
UtilsCheckLink (*класс в vk_maria.responses*), 149, 158
UtilsGetLinkStats (*класс в vk_maria.responses*), 149, 158
UtilsGetShortLink (*класс в vk_maria.responses*), 150, 158
UtilsResolveScreenName (*класс в vk_maria.responses*), 150, 158

V

values() (*метод Adresses*), 197, 250
values() (*метод Career*), 200, 257
values() (*метод City*), 201, 257
values() (*метод Contacts*), 203, 258
values() (*метод Counters*), 206, 259
values() (*метод Country*), 208, 260

values() (*метод Cover*), 210, 251
values() (*метод Currency*), 212, 252
values() (*метод Education*), 214, 261
values() (*метод Images*), 220, 250
values() (*метод LastSeen*), 222, 261
values() (*метод Links*), 224, 252
values() (*метод Market*), 226, 253
values() (*метод MessageInfo*), 184, 185
values() (*метод Military*), 228, 262
values() (*метод Occupation*), 230, 263
values() (*метод Personal*), 234, 264
values() (*метод Place*), 236, 254
values() (*метод Relatives*), 243, 264
values() (*метод Schools*), 246, 265
values() (*метод Universities*), 249, 266
variants_grouping_id (*атрибут Order*), 231, 269
verified (*атрибут Group*), 218, 256
verified (*атрибут GroupsGetById*), 125, 152
verified (*атрибут Profile*), 242, 269
VIDEO_COMMENT_DELETE (*атрибут EventType*), 165, 166
VIDEO_COMMENT_EDIT (*атрибут EventType*), 165, 166
VIDEO_COMMENT_NEW (*атрибут EventType*), 165, 166
VIDEO_COMMENT_RESTORE (*атрибут EventType*), 165, 166
VIDEO_NEW (*атрибут EventType*), 165, 166
videos (*атрибут Counters*), 205, 258
views (*атрибут StoriesGetStats*), 147, 158
Vk (*класс в vk_maria.api*), 16, 43
vk_maria
    модуль, 14
vk_maria.api
    модуль, 15
vk_maria.dispatcher
    модуль, 64
vk_maria.dispatcher.dispatcher
    модуль, 64
vk_maria.dispatcher.filters
    модуль, 67
vk_maria.dispatcher.filters.filters
    модуль, 67
vk_maria.dispatcher.filters.handler
    модуль, 78
vk_maria.dispatcher.fsm
    модуль, 79
vk_maria.dispatcher.fsm.state
    модуль, 80
vk_maria.dispatcher.fsm.storage
    модуль, 82
vk_maria.dispatcher.fsm.storage.core
    модуль, 82
vk_maria.dispatcher.fsm.storage.file
    модуль, 92
vk_maria.dispatcher.fsm.storage.file.base
    модуль, 92
vk_maria.dispatcher.fsm.storage.file.json
    модуль, 97
vk_maria.dispatcher.fsm.storage.file.pickle
    модуль, 102
vk_maria.dispatcher.fsm.storage.memory
    модуль, 107
vk_maria.dispatcher.fsm.storage.memory.memory
    модуль, 107
vk_maria.exceptions
    модуль, 112
vk_maria.longpoll
    модуль, 116
vk_maria.longpoll.longpoll
    модуль, 116
vk_maria.mixins
    модуль, 117
vk_maria.responses
    модуль, 117
vk_maria.types
    модуль, 159
vk_maria.types.callback_query_event
    модуль, 159
vk_maria.types.chat
    модуль, 160
vk_maria.types.event
    модуль, 162
vk_maria.types.event_type
    модуль, 162
vk_maria.types.keyboard
    модуль, 167
vk_maria.types.message
    модуль, 178
vk_maria.types.message_event
    модуль, 187
vk_maria.types.state
    модуль, 189
vk_maria.upload
    модуль, 189
vk_maria.upload.exceptions
    модуль, 189
vk_maria.upload.upload
    модуль, 189
vk_maria.upload.utils
    модуль, 192
vk_maria.utils
    модуль, 193
vk_maria.vk_types
    модуль, 194
VKApps (*атрибут Button*), 169, 177
VKAppsButton (*класс в vk_maria.types.keyboard*), 174, 176
VkMariaException, 114, 116

VKPay (*атрибут Button*), 169, 177
VKPAY_TRANSACTION (*атрибут EventType*), 166, 167
VKPayButton (*класс в vk_maria.types.keyboard*), 175, 176

W

wall (*атрибут Group*), 218, 256
wall (*атрибут GroupsGetById*), 125, 152
wall_close_comments() (*метод Vk*), 43, 64
wall_create_comment() (*метод Vk*), 43, 64
wall_default (*атрибут Profile*), 242, 269
WALL_POST_NEW (*атрибут EventType*), 165, 166
WALL_REPLY_DELETE (*атрибут EventType*), 165, 166
WALL_REPLY_EDIT (*атрибут EventType*), 165, 166
WALL_REPLY_NEW (*атрибут EventType*), 165, 166
WALL_REPLY_RESTORE (*атрибут EventType*), 165, 166
WALL_REPOST (*атрибут EventType*), 165, 166
WallCreateComment (*класс в vk_maria.responses*), 151, 159
width (*атрибут Images*), 219, 250
wiki_page (*атрибут Group*), 218, 256
wiki_page (*атрибут GroupsGetById*), 125, 152
with_traceback() (*метод AccessIsDeniedError*), 112, 115
with_traceback() (*метод AuthorizationError*), 112, 115
with_traceback() (*метод DeprecatedMethodError*), 113, 115
with_traceback() (*метод FiltersFactory.UnknownFilterException*), 72, 78
with_traceback() (*метод InvalidFileFormatError*), 189
with_traceback() (*метод InvalidParametersError*), 113, 115
with_traceback() (*метод KeyIsNotValidError*), 113, 116
with_traceback() (*метод PermissionError*), 113, 115
with_traceback() (*метод ServerError*), 113, 115
with_traceback() (*метод UnknownError*), 114, 115
with_traceback() (*метод UnknownMethodError*), 114, 115
with_traceback() (*метод VkMariaException*), 114, 116
with_traceback() (*метод WrongRequestError*), 114, 115
write() (*метод FileStorage*), 94, 95
write() (*метод JSONStorage*), 98, 100
write() (*метод PickleStorage*), 103, 105
WrongRequestError, 114, 115

Y

year_from (*атрибут Schools*), 245, 264
year_graduated (*атрибут Schools*), 245, 265
year_to (*атрибут Schools*), 245, 265