PyTodoist Documentation

Release 2.1.2

Gary Blackwood

Dec 01, 2018

Contents

CHAPTER 1

PyTodoist

PyTodoist is a Python package for interacting with Todoist. It hides the underlying API calls with higher-level abstractions that make it easy to use Todoist with Python.

1.1 Quick Start

Install the latest version:

```
$ pip install pytodoist
```

Have fun:

```
>>> from pytodoist import todoist
>>> user = todoist.login('gary@garyblackwood.co.uk', 'pa$$w0rd')
>>> projects = user.get_projects()
>>> for project in projects:
...     print(project.name)
...
Inbox
Books to read
Movies to watch
Shopping
Work
Personal
Health
>>> inbox = user.get_project('Inbox')
>>> task = inbox.add_task('Install PyTodoist',
...                       priority=todoist.Priority.VERY_HIGH)
>>> task.complete()
```

1.2 Documentation

Comprehensive online documentation can be found at http://pytodoist.readthedocs.org

CHAPTER 2

Modules

2.1 pytodoist.todoist

This module introduces abstractions over Todoist entities such as Users, Tasks and Projects. It’s purpose is to hide the underlying API calls so that you can interact with Todoist in a straightforward manner.

Example:

```
>>> from pytodoist import todoist
>>> user = todoist.register('John Doe', 'john.doe@gmail.com', 'password')
>>> inbox = user.get_project('Inbox')
>>> install_task = inbox.add_task('Install PyTodoist')
>>> uncompleted_tasks = user.get_uncompleted_tasks()
>>> for task in uncompleted_tasks:
...     print(task.content)
...
Install PyTodoist
>>> install_task.complete()
```

class pytodoist.todoist.**Color**

This class acts as an easy way to specify Todoist project colors.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.add_project('PyTodoist', color=todoist.Color.RED)
```

The supported colors:

- GREEN
- PINK
- LIGHT_ORANGE
- YELLOW

- DARK_BLUE
- BROWN
- PURPLE
- GRAY
- RED
- DARK_ORANGE
- CYAN
- LIGHT_BLUE

class pytodoist.todoist.**Event**

This class acts as an easy way to specify Todoist event types.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.enable_email_notifications(todoist.Event.NOTE_ADDED)
```

The supported events:

- USER_LEFT_PROJECT
- USER_REMOVED_FROM_PROJECT
- ITEM_COMPLETED
- ITEM_UNCOMPLETED
- ITEM_ASSIGNED
- SHARE_INVITATION_REJECTED
- SHARE NOTIFICATION_ACCEPTED
- NOTE_ADDED
- BIZ_TRIAL_WILL_END
- BIZ_TRIAL_ENTER_CC
- BIZ_ACCOUNT_DISABLED
- BIZ_INVITATION_REJECTED
- BIZ_INVITATION_ACCEPTED
- BIZ_PAYMENT_FAILED

class pytodoist.todoist.**Filter**(*filter_json*, *owner*)

A Todoist filter with the following attributes:

Variables

- **id** – The ID of the filter.
- **name** – The filter name.
- ***query*** – The filter query.
- **color** – The color of the filter.
- **item_order** – The order of the filter in the filters list.

- **owner** – The user who owns the label.

delete()

Delete the filter.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> overdue_filter = user.add_filter('Overdue', todoist.Query.OVERDUE)
>>> overdue_filter.delete()
```

update()

Update the filter's details on Todoist.

You must call this method to register any local attribute changes with Todoist.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> overdue_filter = user.add_filter('Overdue', todoist.Query.OVERDUE)
>>> overdue_filter.name = 'OVERDUE!'
... # At this point Todoist still thinks the name is 'Overdue'.
>>> overdue_filter.update()
... # Now the name has been updated on Todoist.
```

class pytodoist.todoist.**Label**(*label_json*, *owner*)

A Todoist label with the following attributes:

Variables

- **id** – The ID of the label.
- **name** – The label name.
- **color** – The color of the label.
- **owner** – The user who owns the label.
- **is_deleted** – Has the label been deleted?

Warning: Requires Todoist premium.

delete()

Delete the label.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> label = user.add_label('family')
>>> label.delete()
```

update()

Update the label's details on Todoist.

You must call this method to register any local attribute changes with Todoist.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> label = user.add_label('family')
>>> label.name = 'friends'
... # At this point Todoist still thinks the name is 'family'.
```

(continues on next page)

(continued from previous page)

```
>>> label.update()
... # Now the name has been updated on Todoist.
```

class pytodoist.todoist.**Note**(*note_json*, *task*)

A Todoist note with the following attributes:

Variables

- **id** – The note ID.
- **content** – The note content.
- **item_id** – The ID of the task it is attached to.
- **task** – The task it is attached to.
- **posted** – The date/time the note was posted.
- **is_deleted** – Has the note been deleted?
- **is_archived** – Has the note been archived?
- **posted_uid** – The ID of the user who attached the note.
- **uids_to_notify** – List of user IDs to notify.

delete()

Delete the note, removing it from it’s task.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> task = project.add_task('Install PyTodoist.')
>>> note = task.add_note('https://pypi.python.org/pypi')
>>> note.delete()
>>> notes = task.get_notes()
>>> print(len(notes))
0
```

update()

Update the note’s details on Todoist.

You must call this method to register any local attribute changes with Todoist.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('Homework')
>>> task = project.add_task('Install PyTodoist.')
>>> note = task.add_note('https://pypi.python.org/pypi')
>>> note.content = 'https://pypi.python.org/pypi/pytodoist'
... # At this point Todoist still thinks the content is the old URL.
>>> note.update()
... # Now the content has been updated on Todoist.
```

class pytodoist.todoist.**Priority**

This class acts as an easy way to specify Todoist task priority.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> inbox = user.get_project('Inbox')
>>> inbox.add_task('Install PyTodoist', priority=todoist.Priority.HIGH)
```

The supported priorities:

- NO
- LOW
- NORMAL
- HIGH
- VERY_HIGH

class `pytodoist.todoist.`**`Project`**(*project_json*, *owner*)

A Todoist Project with the following attributes:

Variables

- **`id`** – The ID of the project.
- **`name`** – The name of the project.
- **`color`** – The color of the project.
- **`collapsed`** – Is this project collapsed?
- **`owner`** – The owner of the project.
- **`last_updated`** – When the project was last updated.
- **`cache_count`** – The cache count of the project.
- **`item_order`** – The task ordering.
- **`indent`** – The indentation level of the project.
- **`is_deleted`** – Has this project been deleted?
- **`is_archived`** – Is this project archived?
- **`inbox_project`** – Is this project the Inbox?

`add_note`(*content*)

Add a note to the project.

Warning: Requires Todoist premium.

Parameters **`content`** (*`str`*) – The note content.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> project.add_note('Remember to update to the latest version.')
```

`add_task`(*content*, *date=None*, *priority=None*)

Add a task to the project

Parameters

- **`content`** (*`str`*) – The task description.
- **`date`** (*`str`*) – The task deadline.
- **`priority`** (*`int`*) – The priority of the task.

Returns The added task.

Return type *pytodoist.todoist.Task*

Note: See here for possible date strings.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> task = project.add_task('Install PyTodoist')
>>> print(task.content)
Install PyTodoist
```

archive()

Archive the project.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> project.archive()
```

collapse()

Collapse the project on Todoist.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> project.collapse()
```

delete()

Delete the project.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> project.delete()
```

delete_collaborator(*email*)

Remove a collaborating user from the shared project.

Parameters **email** (*str*) – The collaborator's email address.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> project.delete_collaborator('jane.doe@gmail.com')
```

get_completed_tasks()

Return a list of all completed tasks in this project.

Returns A list of all completed tasks in this project.

Return type list of *pytodoist.todoist.Task*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
```

(continues on next page)

(continued from previous page)

```
>>> task = project.add_task('Install PyTodoist')
>>> task.complete()
>>> completed_tasks = project.get_completed_tasks()
>>> for task in completed_tasks:
...     task.uncomplete()
```

get_notes()

Return a list of all of the project's notes.

Returns A list of notes.

Return type list of *pytodoist.todoist.Note*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> notes = project.get_notes()
```

get_tasks()

Return all tasks in this project.

Returns A list of all tasks in this project.class

Return type list of *pytodoist.todoist.Task*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> project.add_task('Install PyTodoist')
>>> project.add_task('Have fun!')
>>> tasks = project.get_tasks()
>>> for task in tasks:
...     print(task.content)
Install PyTodoist
Have fun!
```

get_uncompleted_tasks()

Return a list of all uncompleted tasks in this project.

Warning: Requires Todoist premium.

Returns A list of all uncompleted tasks in this project.

Return type list of *pytodoist.todoist.Task*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> project.add_task('Install PyTodoist')
>>> uncompleted_tasks = project.get_uncompleted_tasks()
>>> for task in uncompleted_tasks:
...     task.complete()
```

share(*email*, *message=None*)

Share the project with another Todoist user.

Parameters

- **email** (*str*) – The other user's email address.
- **message** (*str*) – Optional message to send with the invitation.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> project.share('jane.doe@gmail.com')
```

take_ownership()

Take ownership of the shared project.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> project.take_ownership()
```

unarchive()

Unarchive the project.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> project.unarchive()
```

update()

Update the project's details on Todoist.

You must call this method to register any local attribute changes with Todoist.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> project.name = 'Homework'
... # At this point Todoist still thinks the name is 'PyTodoist'.
>>> project.update()
... # Now the name has been updated on Todoist.
```

class pytodoist.todoist.**Query**

This class acts as an easy way to specify search queries.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> tasks = user.search_tasks(todoist.Query.TOMORROW,
...                          todoist.Query.SUNDAY)
```

The supported queries:

- ALL
- TODAY
- TOMORROW
- MONDAY
- TUESDAY
- WEDNESDAY
- THURSDAY

- FRIDAY
- SATURDAY
- SUNDAY
- NO_DUE_DATE
- OVERDUE
- PRIORITY_1
- PRIORITY_2
- PRIORITY_3

class `pytodoist.todoist.`**`Reminder`**(*reminder_json*, *task*)

A Todoist reminder with the following attributes:

Variables

- **`id`** – The ID of the filter.
- **`item_id`** – The ID of the associated task.
- **`service`** – `` `email` ``, `` `sms` `` or `` `push` `` for mobile.
- **`due_date_utc`** – The due date in UTC.
- **`date_string`** – The due date in free form text e.g. `` `every day @ 10` ``
- **`date_lang`** – The language of the date_string.
- **`notify_uid`** – The ID of the user who should be notified.
- **`task`** – The task associated with the reminder.

`delete`()

Delete the reminder.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> task = project.add_task('Install PyTodoist')
>>> task.add_date_reminder('email', '2015-12-01T09:00')
>>> for reminder in task.get_reminders():
...     reminder.delete()
```

exception `pytodoist.todoist.`**`RequestError`**(*response*)

Will be raised whenever a Todoist API call fails.

class `pytodoist.todoist.`**`Task`**(*task_json*, *project*)

A Todoist Task with the following attributes:

Variables

- **`id`** – The task ID.
- **`content`** – The task content.
- **`due_date_utc`** – When is the task due (in UTC).
- **`date_string`** – How did the user enter the task? Could be every day or every day @ 10. The time should be shown when formating the date if @ OR at is found anywhere in the string.
- **`project`** – The parent project.

- **project_id** – The ID of the parent project.
- **checked** – Is the task checked?
- **priority** – The task priority.
- **is_archived** – Is the task archived?
- **indent** – The task indentation level.
- **labels** – A list of attached label names.
- **sync_id** – The task sync ID.
- **in_history** – Is the task in the task history?
- **user_id** – The ID of the user who owns the task.
- **date_added** – The date the task was added.
- **children** – A list of child tasks.
- **item_order** – The task order.
- **collapsed** – Is the task collapsed?
- **has_notifications** – Does the task have notifications?
- **is_deleted** – Has the task been deleted?
- **assigned_by_uid** – ID of the user who assigned the task.
- **responsible_uid** – ID of the user who responsible for the task.

add_date_reminder (*service*, *due_date*)
Add a reminder to the task which activates on a given date.

Warning: Requires Todoist premium.

Parameters

- **service** (*str*) – `email`, `sms` or `push` for mobile.
- **due_date** (*str*) – The due date in UTC, formatted as `YYYY-MM-DDTHH:MM`

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> task = project.add_task('Install PyTodoist')
>>> task.add_date_reminder('email', '2015-12-01T09:00')
```

add_location_reminder (*service*, *name*, *lat*, *long*, *trigger*, *radius*)
Add a reminder to the task which activates on at a given location.

Warning: Requires Todoist premium.

Parameters

- **service** (*str*) – `email`, `sms` or `push` for mobile.
- **name** (*str*) – An alias for the location.

- **lat** (*float*) – The location latitude.
- **long** (*float*) – The location longitude.
- **trigger** (*str*) – `on_enter` or `on_leave`.
- **radius** (*float*) – The radius around the location that is still considered the location.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> task = project.add_task('Install PyTodoist')
>>> task.add_location_reminder('email', 'Leave Glasgow',
...                            55.8580, 4.2590, 'on_leave', 100)
```

add_note (*content*)
Add a note to the Task.

Warning: Requires Todoist premium.

Parameters **content** (*str*) – The content of the note.

Returns The added note.

Return type *pytodoist.todoist.Note*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> task = project.add_task('Install Todoist.')
>>> note = task.add_note('https://pypi.python.org/pypi')
>>> print(note.content)
https://pypi.python.org/pypi
```

complete ()
Mark the task complete.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> task = project.add_task('Install PyTodoist')
>>> task.complete()
```

delete ()
Delete the task.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('Homework')
>>> task = project.add_task('Read Chapter 4')
>>> task.delete()
```

get_notes ()
Return all notes attached to this Task.

Returns A list of all notes attached to this Task.

Return type list of *pytodoist.todoist.Note*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> task = project.add_task('Install PyTodoist.')
>>> task.add_note('https://pypi.python.org/pypi')
>>> notes = task.get_notes()
>>> print(len(notes))
1
```

get_reminders()

Return a list of the task's reminders.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> task = project.add_task('Install PyTodoist')
>>> task.add_date_reminder('email', '2015-12-01T09:00')
>>> reminders = task.get_reminders()
```

move(*project*)

Move this task to another project.

Parameters **project** (*pytodoist.todoist.Project*) – The project to move the task to.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> task = project.add_task('Install PyTodoist')
>>> print(task.project.name)
PyTodoist
>>> inbox = user.get_project('Inbox')
>>> task.move(inbox)
>>> print(task.project.name)
Inbox
```

uncomplete()

Mark the task uncomplete.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> task = project.add_task('Install PyTodoist')
>>> task.uncomplete()
```

update()

Update the task's details on Todoist.

You must call this method to register any local attribute changes with Todoist.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> task = project.add_task('Install PyTodoist')
>>> task.content = 'Install the latest version of PyTodoist'
... # At this point Todoist still thinks the content is
... # 'Install PyTodoist'
```

(continues on next page)

(continued from previous page)

```
>>> task.update()
... # Now the content has been updated on Todoist.
```

class pytodoist.todoist.**TodoistObject**(*object_json*)
A helper class which 'converts' a JSON object into a python object.

class pytodoist.todoist.**User**(*user_json*)
A Todoist User that has the following attributes:

Variables

- **id** – The ID of the user.
- **email** – The user's email address.
- **password** – The user's password.
- **full_name** – The user's full name.
- **join_date** – The date the user joined Todoist.
- **is_premium** – Does the user have Todoist premium?
- **premium_until** – The date on which the premium status is revoked.
- **tz_info** – The user's timezone information.
- **time_format** – The user's selected time_format. If `0` then show time as `13:00` otherwise `1pm`.
- **date_format** – The user's selected date format. If `0` show dates as `DD-MM-YYY` otherwise `MM-DD-YYYY`.
- **start_page** – The new start page. `_blank`: for a blank page, `_info_page` for the info page, `_project_$PROJECT_ID` for a project page or `$ANY_QUERY` to show query results.
- **start_day** – The new first day of the week `(1-7, Mon-Sun)`.
- **next_week** – The day to use when postponing `(1-7, Mon-Sun)`.
- **sort_order** – The user's sort order. If it's `0` then show the oldest dates first when viewing projects, otherwise oldest dates last.
- **mobile_number** – The user's mobile number.
- **mobile_host** – The host of the user's mobile.
- **business_account_id** – The ID of the user's business account.
- **karma** – The user's karma.
- **karma_trend** – The user's karma trend.
- **default_reminder** – `email` for email, `mobile` for SMS, `push` for smart device notifications or `no_default` to turn off notifications. Only for premium users.
- **inbox_project** – The ID of the user's Inbox project.
- **team_inbox** – The ID of the user's team Inbox project.
- **api_token** – The user's API token.
- **shard_id** – The user's shard ID.
- **image_id** – The ID of the user's avatar.

- **is_biz_admin** – Is the user a business administrator?
- **last_used_ip** – The IP address of the computer last used to login.
- **auto_reminder** – The auto reminder of the user.

add_filter (*name*, *query*, *color=None*, *item_order=None*)
Create a new filter.

> **Warning:** Requires Todoist premium.

Parameters

- **name** – The name of the filter.
- **query** – The query to search for.
- **color** – The color of the filter.
- **item_order** – The filter's order in the filter list.

Returns The newly created filter.

Return type *pytodoist.todoist.Filter*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> overdue_filter = user.add_filter('Overdue', todoist.Query.OVERDUE)
```

add_label (*name*, *color=None*)
Create a new label.

> **Warning:** Requires Todoist premium.

Parameters

- **name** (*str*) – The name of the label.
- **color** (*str*) – The color of the label.

Returns The newly created label.

Return type *pytodoist.todoist.Label*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> label = user.add_label('family')
```

add_project (*name*, *color=None*, *indent=None*, *order=None*)
Add a project to the user's account.

Parameters **name** (*str*) – The project name.

Returns The project that was added.

Return type *pytodoist.todoist.Project*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.add_project('PyTodoist')
>>> print(project.name)
PyTodoist
```

clear_reminder_locations()
Clear all reminder locations set for the user.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.clear_reminder_locations()
```

delete(*reason=None*)
Delete the user's account from Todoist.

Warning: You cannot recover the user after deletion!

Parameters **reason** (*str*) – The reason for deletion.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.delete()
... # The user token is now invalid and Todoist operations will fail.
```

disable_email_notifications(*event*)
Disable email notifications for a given event.

Parameters **event** (*str*) – The event to disable email notifications for.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.disable_email_notifications(todoist.Event.NOTE_ADDED)
```

disable_karma()
Disable karma for the user.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.disable_karma()
```

disable_push_notifications(*event*)
Disable push notifications for a given event.

Parameters **event** (*str*) – The event to disable push notifications for.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.disable_push_notifications(todoist.Event.NOTE_ADDED)
```

disable_vacation()
Disable vacation for the user.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.disable_vacation()
```

enable_email_notifications(*event*)
Enable email notifications for a given event.

Parameters **event** (*str*) – The event to enable email notifications for.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.enable_email_notifications(todoist.Event.NOTE_ADDED)
```

enable_karma()
Enable karma for the user.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.enable_karma()
```

enable_push_notifications(*event*)
Enable push notifications for a given event.

Parameters **event** (*str*) – The event to enable push notifications for.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.enable_push_notifications(todoist.Event.NOTE_ADDED)
```

enable_vacation()
Enable vacation for the user.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.enable_vacation()
```

get_archived_projects()
Return a list of a user's archived projects.

Returns The user's archived projects.

Return type list of *pytodoist.todoist.Project*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('PyTodoist')
>>> project.archive()
>>> projects = user.get_archived_projects()
>>> for project in projects:
...     print(project.name)
PyTodoist
```

get_completed_tasks()
Return all of a user's completed tasks.

Warning: Requires Todoist premium.

Returns A list of completed tasks.

Return type list of *pytodoist.todoist.Task*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> completed_tasks = user.get_completed_tasks()
>>> for task in completed_tasks:
...     task.uncomplete()
```

get_filter(*name*)
Return the filter that has the given filter name.

Parameters **name** – The name to search for.

Returns The filter with the given name.

Return type *pytodoist.todoist.Filter*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.add_filter('Overdue', todoist.Query.OVERDUE)
>>> overdue_filter = user.get_filter('Overdue')
```

get_filters()
Return a list of all a user's filters.

Returns A list of filters.

Return type list of *pytodoist.todoist.Filter*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> filters = user.get_filters()
```

get_label(*label_name*)
Return the user's label that has a given name.

Parameters **label_name** (*str*) – The name to search for.

Returns A label that has a matching name or `None` if not found.

Return type *pytodoist.todoist.Label*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> label = user.get_label('family')
```

get_labels()
Return a list of all of a user's labels.

Returns A list of labels.

Return type list of *pytodoist.todoist.Label*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> labels = user.get_labels()
```

get_notes()
Return a list of all of a user's notes.

Returns A list of notes.

Return type list of *pytodoist.todoist.Note*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> notes = user.get_notes()
```

get_productivity_stats()

Return the user's productivity stats.

Returns A JSON-encoded representation of the user's productivity stats.

Return type A JSON-encoded object.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> stats = user.get_productivity_stats()
>>> print(stats)
{"karma_last_update": 50.0, "karma_trend": "up", ... }
```

get_project(*project_name*)

Return the project with a given name.

Parameters **project_name** (*str*) – The name to search for.

Returns The project that has the name `project_name` or `None` if no project is found.

Return type *pytodoist.todoist.Project*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> project = user.get_project('Inbox')
>>> print(project.name)
Inbox
```

get_projects()

Return a list of a user's projects.

Returns The user's projects.

Return type list of *pytodoist.todoist.Project*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.add_project('PyTodoist')
>>> projects = user.get_projects()
>>> for project in projects:
...     print(project.name)
Inbox
PyTodoist
```

get_redirect_link()

Return the absolute URL to redirect or to open in a browser. The first time the link is used it logs in the user automatically and performs a redirect to a given page. Once used, the link keeps working as a plain redirect.

Returns The user's redirect link.

Return type str

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> print(user.get_redirect_link())
https://todoist.com/secureRedirect?path=%2Fapp&token ...
```

get_reminders()
Return a list of the user's reminders.

Returns A list of reminders.

Return type list of *pytodoist.todoist.Reminder*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> reminders = user.get_reminders()
```

get_tasks()
Return all of a user's tasks, regardless of completion state.

Returns A list of all of a user's tasks.

Return type list of *pytodoist.todoist.Task*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> tasks = user.get_tasks()
```

get_uncompleted_tasks()
Return all of a user's uncompleted tasks.

Warning: Requires Todoist premium.

Returns A list of uncompleted tasks.

Return type list of *pytodoist.todoist.Task*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> uncompleted_tasks = user.get_uncompleted_tasks()
>>> for task in uncompleted_tasks:
...     task.complete()
```

search_tasks(**queries*)
Return a list of tasks that match some search criteria.

Note: Example queries can be found here.

Note: A standard set of queries are available in the *pytodoist.todoist.Query* class.

Parameters **queries** (*list str*) – Return tasks that match at least one of these queries.

Returns A list tasks that match at least one query.

Return type list of *pytodoist.todoist.Task*

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> tasks = user.search_tasks(todoist.Query.TOMORROW, '18 Sep')
```

sync (*resource_types='["all"]'*)

Synchronize the user's data with the Todoist server.

This function will pull data from the Todoist server and update the state of the user object such that they match. It does not *push* data to Todoist. If you want to do that use `pytodoist.todoist.User.update()`.

Parameters **resource_types** – A JSON-encoded list of Todoist resources which should be synced. By default this is everything, but you can choose to sync only selected resources. See here for a list of resources.

update ()

Update the user's details on Todoist.

This method must be called to register any local attribute changes with Todoist.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.full_name = 'John Smith'
>>> # At this point Todoist still thinks the name is 'John Doe'.
>>> user.update()
>>> # Now the name has been updated on Todoist.
```

update_daily_karma_goal (*goal*)

Update the user's daily karma goal.

Parameters **goal** (*int*) – The daily karma goal.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.update_daily_karma_goal(100)
```

update_weekly_karma_goal (*goal*)

Set the user's weekly karma goal.

Parameters **goal** (*int*) – The weekly karma goal.

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
>>> user.update_weekly_karma_goal(700)
```

pytodoist.todoist.**login** (*email*, *password*)

Login to Todoist.

Parameters

- **email** (*str*) – A Todoist user's email address.
- **password** (*str*) – A Todoist user's password.

Returns The Todoist user.

Return type `pytodoist.todoist.User`

```
>>> from pytodoist import todoist
>>> user = todoist.login('john.doe@gmail.com', 'password')
```

(continues on next page)

(continued from previous page)

```
>>> print(user.full_name)
John Doe
```

pytodoist.todoist.**login_with_api_token**(*api_token*)

Login to Todoist using a user's api token.

Note: It is up to you to obtain the api token.

Parameters **api_token** (*str*) – A Todoist user's api token.

Returns The Todoist user.

Return type *pytodoist.todoist.User*

```
>>> from pytodoist import todoist
>>> api_token = 'api_token'
>>> user = todoist.login_with_api_token(api_token)
>>> print(user.full_name)
John Doe
```

pytodoist.todoist.**login_with_google**(*email*, *oauth2_token*)

Login to Todoist using Google oauth2 authentication.

Parameters

- **email** (*str*) – A Todoist user's Google email address.
- **oauth2_token** (*str*) – The oauth2 token associated with the email.

Returns The Todoist user.

Return type *pytodoist.todoist.User*

Note: It is up to you to obtain the valid oauth2 token.

```
>>> from pytodoist import todoist
>>> oauth2_token = 'oauth2_token'
>>> user = todoist.login_with_google('john.doe@gmail.com', oauth2_token)
>>> print(user.full_name)
John Doe
```

pytodoist.todoist.**register**(*full_name*, *email*, *password*, *lang=None*, *timezone=None*)

Register a new Todoist account.

Parameters

- **full_name** (*str*) – The user's full name.
- **email** (*str*) – The user's email address.
- **password** (*str*) – The user's password.
- **lang** (*str*) – The user's language.
- **timezone** (*str*) – The user's timezone.

Returns The Todoist user.

Return type *pytodoist.todoist.User*

```
>>> from pytodoist import todoist
>>> user = todoist.register('John Doe', 'john.doe@gmail.com', 'password')
>>> print(user.full_name)
John Doe
```

pytodoist.todoist.**register_with_google**(*full_name*, *email*, *oauth2_token*, *lang=None*, *timezone=None*)

Register a new Todoist account by linking a Google account.

Parameters

- **full_name** (*str*) – The user's full name.
- **email** (*str*) – The user's email address.
- **oauth2_token** (*str*) – The oauth2 token associated with the email.
- **lang** (*str*) – The user's language.
- **timezone** (*str*) – The user's timezone.

Returns The Todoist user.

Return type *pytodoist.todoist.User*

Note: It is up to you to obtain the valid oauth2 token.

```
>>> from pytodoist import todoist
>>> oauth2_token = 'oauth2_token'
>>> user = todoist.register_with_google('John Doe', 'john.doe@gmail.com',
...                                          oauth2_token)
>>> print(user.full_name)
John Doe
```

2.2 pytodoist.api

This module is a pure wrapper around version 6 of the Todoist API.

If you do not need access to the raw HTTP response to the request, consider using the higher level abstractions implemented in the *pytodoist.todoist* module.

Example:

```
>>> from pytodoist.api import TodoistAPI
>>> api = TodoistAPI()
>>> response = api.login('john.doe@gmail.com', 'password')
>>> user_info = response.json()
>>> full_name = user_info['full_name']
>>> print(full_name)
John Doe
```

class pytodoist.api.**TodoistAPI**

A wrapper around version 7 of the Todoist API.

```
>>> from pytodoist.api import TodoistAPI
>>> api = TodoistAPI()
>>> print(api.URL)
https://api.todoist.com/API/v7/
```

add_item(*api_token*, *content*, ***kwargs*)
Add a task to a project.

Parameters

- **token** (*str*) – The user's login token.
- **content** (*str*) – The task description.
- **project_id** (*str*) – The project to add the task to. Default is `Inbox`
- **date_string** (*str*) – The deadline date for the task.
- **priority** (*int*) – The task priority `(1-4)`.
- **indent** (*int*) – The task indentation `(1-4)`.
- **item_order** (*int*) – The task order.
- **children** (*str*) – A list of child tasks IDs.
- **labels** (*str*) – A list of label IDs.
- **assigned_by_uid** (*str*) – The ID of the user who assigns current task. Accepts 0 or any user id from the list of project collaborators. If value is unset or invalid it will automatically be set up by your uid.
- **responsible_uid** (*str*) – The id of user who is responsible for accomplishing the current task. Accepts 0 or any user id from the list of project collaborators. If the value is unset or invalid it will automatically be set to null.
- **note** (*str*) – Content of a note to add.

Returns The HTTP response to the request.

Return type `requests.Response`

```
>>> from pytodoist.api import TodoistAPI
>>> api = TodoistAPI()
>>> response = api.login('john.doe@gmail.com', 'password')
>>> user_info = response.json()
>>> user_api_token = user_info['token']
>>> response = api.add_item(user_api_token, 'Install PyTodoist')
>>> task = response.json()
>>> print(task['content'])
Install PyTodoist
```

delete_user(*api_token*, *password*, ***kwargs*)
Delete a registered Todoist user's account.

Parameters

- **api_token** (*str*) – The user's login api_token.
- **password** (*str*) – The user's password.
- **reason_for_delete** (*str*) – The reason for deletion.
- **in_background** (*int*) – If `0`, delete the user instantly.

Returns The HTTP response to the request.

Return type `requests.Response`

`get_all_completed_tasks`(*api_token*, ***kwargs*)

Return a list of a user's completed tasks.

> **Warning:** Requires Todoist premium.

Parameters

- **`api_token`** (*str*) – The user's login api_token.
- **`project_id`** (*str*) – Filter the tasks by project.
- **`limit`** (*int*) – The maximum number of tasks to return (default `30`, max `50`).
- **`offset`** (*int*) – Used for pagination if there are more tasks than limit.
- **`from_date`** (*str*) – Return tasks with a completion date on or older than from_date. Formatted as `2007-4-29T10:13`.
- **`to`** – Return tasks with a completion date on or less than to_date. Formatted as `2007-4-29T10:13`.

Returns The HTTP response to the request.

Return type `requests.Response`

```
>>> from pytodoist.api import TodoistAPI
>>> api = TodoistAPI()
>>> response = api.login('john.doe@gmail.com', 'password')
>>> user_info = response.json()
>>> user_api_token = user_info['api_token']
>>> response = api.get_all_completed_tasks(user_api_token)
>>> completed_tasks = response.json()
```

`get_productivity_stats`(*api_token*, ***kwargs*)

Return a user's productivity stats.

Parameters **`api_token`** (*str*) – The user's login api_token.

Returns The HTTP response to the request.

Return type `requests.Response`

`get_redirect_link`(*api_token*, ***kwargs*)

Return the absolute URL to redirect or to open in a browser. The first time the link is used it logs in the user automatically and performs a redirect to a given page. Once used, the link keeps working as a plain redirect.

Parameters

- **`api_token`** (*str*) – The user's login api_token.
- **`path`** (*str*) – The path to redirect the user's browser. Default `/app`.
- **`hash`** (*str*) – The has part of the path to redirect the user's browser.

Returns The HTTP response to the request.

Return type `requests.Response`

```
>>> from pytodoist.api import TodoistAPI
>>> api = TodoistAPI()
>>> response = api.login('john.doe@gmail.com', 'password')
>>> user_info = response.json()
>>> user_api_token = user_info['api_token']
>>> response = api.get_redirect_link(user_api_token)
>>> link_info = response.json()
>>> redirect_link = link_info['link']
>>> print(redirect_link)
https://todoist.com/secureRedirect?path=adflk...
```

login(*email*, *password*)

Login to Todoist.

Parameters

- **email** (*str*) – The user's email address.
- **password** (*str*) – The user's password.

Returns The HTTP response to the request.

Return type `requests.Response`

```
>>> from pytodoist.api import TodoistAPI
>>> api = TodoistAPI()
>>> response = api.login('john.doe@gmail.com', 'password')
>>> user_info = response.json()
>>> full_name = user_info['full_name']
>>> print(full_name)
John Doe
```

login_with_google(*email*, *oauth2_token*, ***kwargs*)

Login to Todoist using Google's oauth2 authentication.

Parameters

- **email** (*str*) – The user's Google email address.
- **oauth2_token** (*str*) – The user's Google oauth2 token.
- **auto_signup** (*int*) – If `1` register an account automatically.
- **full_name** (*str*) – The full name to use if the account is registered automatically. If no name is given an email based nickname is used.
- **timezone** (*str*) – The timezone to use if the account is registered automatically. If no timezone is given one is chosen based on the user's IP address.
- **lang** (*str*) – The user's language.

Returns The HTTP response to the request.

Return type `requests.Response`

```
>>> from pytodoist.api import TodoistAPI
>>> api = TodoistAPI()
>>> oauth2_token = 'oauth2_token'  # Get this from Google.
>>> response = api.login_with_google('john.doe@gmail.com',
...                                   oauth2_token)
>>> user_info = response.json()
>>> full_name = user_info['full_name']
```

(continues on next page)

(continued from previous page)

```
>>> print(full_name)
John Doe
```

query (*api_token*, *queries*, ***kwargs*)
Search all of a user's tasks using date, priority and label queries.

Parameters

- **api_token** (*str*) – The user's login api_token.
- **queries** (*list (str)*) – A JSON list of queries to search. See examples here.
- **as_count** (*int*) – If `1` then return the count of matching tasks.

Returns The HTTP response to the request.

Return type `requests.Response`

register (*email*, *full_name*, *password*, ***kwargs*)
Register a new Todoist user.

Parameters

- **email** (*str*) – The user's email.
- **full_name** (*str*) – The user's full name.
- **password** (*str*) – The user's password.
- **lang** (*str*) – The user's language.
- **timezone** (*str*) – The user's timezone.

Returns The HTTP response to the request.

Return type `requests.Response`

```
>>> from pytodoist.api import TodoistAPI
>>> api = TodoistAPI()
>>> response = api.register('john.doe@gmail.com', 'John Doe',
...                         'password')
>>> user_info = response.json()
>>> full_name = user_info['full_name']
>>> print(full_name)
John Doe
```

sync (*api_token*, *sync_token*, *resource_types='["all"]'*, ***kwargs*)
Update and retrieve Todoist data.

Parameters

- **api_token** (*str*) – The user's login api_token.
- **seq_no** (*int*) – The request sequence number. On initial request pass `0`. On all others pass the last seq_no you received.
- **seq_no_global** (*int*) – The request sequence number. On initial request pass `0`. On all others pass the last seq_no you received.
- **resource_types** – Specifies which subset of data you want to receive e.g. only projects. Defaults to all data.
- **commands** (*list (str)*) – A list of JSON commands to perform.

Returns The HTTP response to the request.

Return type `requests.Response`

```
>>> from pytodoist.api import TodoistAPI
>>> api = TodoistAPI()
>>> response = api.register('john.doe@gmail.com', 'John Doe',
...                         'password')
>>> user_info = response.json()
>>> api_token = user_info['api_token']
>>> response = api.sync(api_token, 0, 0, '["projects"]')
>>> print(response.json())
{'seq_no_global': 3848029654, 'seq_no': 3848029654, 'Projects': ...}
```

update_notification_settings (*api_token*, *event*, *service*, *should_notify*)
Update a user's notification settings.

Parameters

- **api_token** (*str*) – The user's login api_token.
- **event** (*str*) – Update the notification settings of this event.
- **service** (*str*) – `email` or `push`
- **should_notify** (*int*) – If `0` notify, otherwise do not.

Returns The HTTP response to the request.

Return type `requests.Response`

```
>>> from pytodoist.api import TodoistAPI
>>> api = TodoistAPI()
>>> response = api.login('john.doe@gmail.com', 'password')
>>> user_info = response.json()
>>> user_api_token = user_info['api_token']
>>> response = api.update_notification_settings(user_api_token,
...                                              'user_left_project',
...                                              'email', 0)
...
```

upload_file (*api_token*, *file_path*, ***kwargs*)
Upload a file suitable to be passed as a file_attachment.

Parameters

- **api_token** (*str*) – The user's login api_token.
- **file_path** (*str*) – The path of the file to be uploaded.

Returns The HTTP response to the request.

Return type `requests.Response`

CHAPTER 3

Contributing

3.1 Reporting Issues

Please report all issues using the github issue tracker. If you would like to ask a question, feel free to send me an email.

3.2 Pull Requests

Please fork the project on github and submit a pull request.

1. Write tests for any new code.
2. Ensure that your code meets the PEP8 standards.
3. Update the documentation.
4. Add yourself to credits.rst

CHAPTER 4

Credits

4.1 Lead Developer

- Gary Blackwood <gary@garyblackwood.co.uk>

4.2 Contributors

- Elias Kuthe <elias.kuthe@web.de>
- Ernie Hershey <github@ernie.org>

CHAPTER 5

History

5.1 2.1.2 (23/11/2018)

- Upgraded the requests dependency to fix a security vulnerability.

5.2 2.1.1 (17/07/2018)

- Improved handling of invalid data in API responses.

5.3 2.1.0 (25/02/2017)

- Migrated from version 6 to version 7 of the sync API.

5.4 2.0.4 (09/12/2015)

- Fixed a bug in Project.get_tasks which was causing all tasks to be returned.

5.5 2.0.3 (05/11/2015)

- Compatibility fixes due to official sync API changes.

5.6 2.0.2 (23/04/2015)

- Fixed a couple of issues caused by 2.0.1.

5.7 2.0.1 (23/04/2015)

- Added seq_no_global to all sync calls as the Todoist API was updated.

5.8 2.0.0 (21/04/2015)

- Added support for the Todoist API v6.
- Added support for wheel distribution.

5.9 1.0.0 (11/09/2014)

- Initial release.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index