Schedule Manager

Release 0.1.1

Aug 12, 2020

Contents

A thread-based task scheduler.
Schedule manager provides an easy way to schedule your jobs.

Periodic, daily, weekly, monthly or even non-periodic jobs are available for scheduling as tasks.

CHAPTER 1

Quick Installation

```
$ pip install schedule-manager
```

CHAPTER 2

Example

```
from schedule_manager import ScheduleManager
from datetime import datetime


def jobs():
    current = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    print("Working now {}".format(current))


manager = ScheduleManager()

# Schedule a periodic task: do job every 60 seconds
manager.register_task(name="task1", job=jobs).period(60).start()

# Schedule a daily task: do job at 18:00 every day
manager.register_task(name="task2",
                      job=jobs).period_day_at("18:00:00").start()

# Schedule a periodic task: start task at 21:00
manager.register_task(name="task3",
                      job=jobs).period(90).start_at("21:00:00").start()

# Schedule a non-periodic task: do job 5 times
manager.register_task(name="task4",
                      job=jobs).period(30).nonperiodic(5).start()


# Pause task1
manager.task("task1").pause()



# Stop all tasks
manager.all_tasks.stop()
```

CHAPTER 3

User Guide

3.1 Installation

This part of documentation will introduce how to install schedule manager.

You are able to install schedule manager using pip or from source code.

3.1.1 Using pip

Install schedule manager with following command:

```
$ pip install schedule-manager
```

If you do not have pip installed, install it this first.

3.1.2 From Source Code

It is able to get source code from GitHub.

Or you may clone the repository with git:

```
$ git clone git://github.com/e619003/ScheduleManager.git
```

Once you have downloaded the source code, you can install it using pip:

```
$ cd ScheduleManager
$ pip install .
```

Or install it with setup.py:

```
$ cd ScheduleManager
$ python setup.py install
```

3.2 Quick Start

Schedule Manager provides an easy way to schedule your jobs. It allows people to creates and schedule tasks to execute jobs in particular time.

This part of documentation will introduce how to use Schedule Manager.

Please make sure Schedule Manager is *installed* first.

3.2.1 Task Management

Task object is used for scheduling a job to be run as the particular time.

Although *Task* object is able to be used directly, *ScheduleManager* object provides a convenient way to manage multiple *Task* objects.

Task Registration

ScheduleManager object is used to manage tasks.

It provides two ways to let Task can be managed by *ScheduleManager*.

1. With `register_task` method:

Using *register_task* method is able to create a new task and register it in manger directly.

```
from schedule_manager import ScheduleManager


def jobs():
    current = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    print("Working now {}".format(current))


manager = ScheduleManager()

# Register job as a new task
manager.register_task(name="task", job=jobs).period(60).start()
```

2. With `register` method:

Use *register* method to register an exist task.

```
from schedule_manager import ScheduleManager
from schedule_manager import Task


manager = ScheduleManager()

task = Task(name="task", job=print, args=("Hello task",)).period(60)
task.start()
```

(continues on next page)

(continued from previous page)

```
# Register an existing task
manager.register(task)
```

Tagging Tasks

Task object is able to be identified by tags.

We can label Task with different tags for *better management*.

1. Add tags

A tag can be any `Object` like `str`, `int`, and so on.

We can use *add_tag* method to add single tag or use *add_tags* method to add multiple tags to the task.

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",)).period(60)


# Add tag "tag1"
task.add_tag("tag1")

# Add tags "tagA" and "label-A"
task.add_tags(["tagA", "label-A"])
```

2. Remove tags

Tags are able to be removed if we do not need them.

We can use *remove_tag* method to remove single tag or use *remove_tags* method to remove multiple tags to the task.

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",)).period(60)
task.add_tags(["tagA", "label-A", "event-1", "job-I"])


# Remove tag "tagA"
task.remove_tag("tagA")

# Remove tags "label-A" and "job-I"
task.remove_tags(["label-A", "job-I"])
```

3. Set specific tags

We can use *set_tags* method set specific tags to the task.

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",)).period(60)
task.add_tags(["tagA", "label-A", "event-1", "job-I"])


# Set tags to "tagB" and "tag2"
# Now tags of the task will be changed to "tagB" and "tag2"
task.set_tags(["tagB", "tag2"])
```

4. View tags of a task

A task is able to have lots of tags in its tag list.
We are able to use *tag* property to list all tags of a task.

```
>>> from schedule_manager import Task
>>> task = Task(name="task", job=print, args=("Hello task",))
>>> task.set_tags(["tag1", "tagA"]).add_tag("label-A")
Task<(task, initial daemon, None)>
>>> task.tag    # List tags
['tag1', 'tagA', 'label-A']
```

Remove Task from ScheduleManager

If we do not want to manaage specific task with *ScheduleManager*, we are able to remove the task from the manager.

ScheduleManager object provides *unregister* method to remove the task from itself. We can use *unregister* method to unregister tasks *by name* or *by tags*.

1. Unregister task by name

Here shows how to unregister a task by name.

```
from schedule_manager import ScheduleManager
from schedule_manager import Task

manager = ScheduleManager()

task1 = Task(name="task1", job=print, args=("Hello task1",)).period(60)
manager.register(task1)

task2 = Task(name="task2", job=print, args=("Hello task2",)).period(60)
manager.register(task2)


# Unregister task by name.
# task1 will be removed
manager.unregister(name="task1")
```

2. Unregister tasks by tag

Here shows how to unregister tasks by a tag.

```
from schedule_manager import ScheduleManager
from schedule_manager import Task

manager = ScheduleManager()

task1 = Task(name="task1", job=print, args=("Hello task1",)).period(60)
task1.add_tag("tag-1")
manager.register(task1)

task2 = Task(name="task2", job=print, args=("Hello task2",)).period(60)
task2.add_tag("tag-2")
manager.register(task2)


# Unregister task by tag.
# task1 will be removed
manager.unregister(tag="tag-1")
```

3. Unregister tasks by tags

Here shows how to unregister tasks by several tags.

```
from schedule_manager import ScheduleManager
from schedule_manager import Task

manager = ScheduleManager()

task1 = Task(name="task1", job=print, args=("Hello task1",)).period(60)
task1.add_tag("tag-1")
manager.register(task1)

task2 = Task(name="task2", job=print, args=("Hello task2",)).period(60)
task2.add_tag("tag-2")
manager.register(task2)

task3 = Task(name="task3", job=print, args=("Hello task3",)).period(60)
task3.add_tag("tag-I")
manager.register(task3)

task4 = Task(name="task4", job=print, args=("Hello task4",)).period(60)
task4.add_tag("tag-I")
manager.register(task4)


# Unregister task by tags.
# task1, task3 and task4 will be removed
manager.unregister(tag=["tag-1", "tag-I"])
```

Managing Registered Tasks

Tasks can be configured before and after being registerd in a *ScheduleManager*.

Once we want to configure registered tasks, we need to obtain them from schedule manager first.

1. Obtain task by task name

Tasks with same name are not able to register in a *ScheduleManager* at the same time.

Every task have an unique name. We can obtain specific task by searching with the name with *task* method.

```
from schedule_manager import ScheduleManager

manager = ScheduleManager()

manager.register_task(name="task1", job=print, args=("Hello task1",))
manager.register_task(name="task2", job=print, args=("Hello task2",))

# Obtain task by task name.
# We will get task named 'task1'
task = manager.task("task1")

# Task is able to be configured directly after obtaining it.
# Get task named 'task1' and add a tag 'tag-I' to it
manager.task("task1").add_tag("tag-I")
```

2. Obtain tasks by task tags

Tasks are able to be *identified by tags*. We can obtain tasks by searching with the tags with *tasks* method.

```
from schedule_manager import ScheduleManager

manager = ScheduleManager()

manager.register_task(name="task1", job=print, args=("Hello task1",))
manager.task("task1").add_tag("type-I")
manager.register_task(name="task2", job=print, args=("Hello task2",))
manager.task("task2").add_tag("type-II")
manager.register_task(name="task3", job=print, args=("Hello task3",))
manager.task("task3").add_tag("type-I")
manager.register_task(name="task4", job=print, args=("Hello task4",))
manager.task("task4").add_tag("type-III")

# Obtain tasks by tags.
# We will get tasks named 'task1' and 'task3'
# Note: return will be a TaskGroup instance which can be operated
#       like a Task instance
task_group = manager.tasks("type-I")

# This will get tasks named 'task1', 'task3' and 'task4'
task_group2 = manager.tasks(["type-I", "type-III"])

# Tasks are able to be configured directly after obtaining them.
# Get tasks and add a tag 'tag-I' to them
manager.tasks("type-I").add_tag("tag-I")
```

3. Obtain all tasks registered in the schedule manager

It is able to obtain all tasks registered in the schedule manager with *all_tasks* property.

```
from schedule_manager import ScheduleManager

manager = ScheduleManager()

manager.register_task(name="task1", job=print, args=("Hello task1",))
manager.register_task(name="task2", job=print, args=("Hello task2",))
manager.register_task(name="task3", job=print, args=("Hello task3",))
manager.register_task(name="task4", job=print, args=("Hello task4",))

# Obtain all tasks registered in the schedule manager.
# Note: return will be a TaskGroup instance which can be operated
#       like a Task instance
task_group = manager.all_tasks

# Tasks are able to be configured directly after obtaining them.
# Get tasks and add a tag 'tag-I' to them
manager.all_tasks.add_tag("tag-I")
```

4. Obtain all running tasks registered in the schedule manager

It is able to obtain all running tasks registered in the schedule manager with *running_tasks* property.

```
from schedule_manager import ScheduleManager

manager = ScheduleManager()

manager.register_task(name="task1", job=print, args=("Hello task1",))
manager.task("task1").period(10).start()
manager.register_task(name="task2", job=print, args=("Hello task2",))
manager.register_task(name="task3", job=print, args=("Hello task3",))
manager.register_task(name="task4", job=print, args=("Hello task4",))
manager.task("task4").period(30).start()

# Obtain tasks named 'task1' and 'task4'
# Note: return will be a TaskGroup instance which can be operated
#       like a Task instance
task_group = manager.running_tasks

# Tasks are able to be configured directly after obtaining them.
# Get tasks and add a tag 'tag-I' to them
manager.running_tasks.add_tag("tag-I")
```

5. Obtain all pending tasks registered in the schedule manager

It is able to obtain all pending tasks registered in the schedule manager with *pending_tasks* property.

```
from schedule_manager import ScheduleManager

manager = ScheduleManager()

```

(continues on next page)

(continued from previous page)

```
manager.register_task(name="task1", job=print, args=("Hello task1",))
manager.task("task1").period(10).start()
manager.register_task(name="task2", job=print, args=("Hello task2",))
manager.register_task(name="task3", job=print, args=("Hello task3",))
manager.register_task(name="task4", job=print, args=("Hello task4",))
manager.task("task4").period(30).start()

# Obtain tasks named 'task2' and 'task3'
# Note: return will be a TaskGroup instance which can be operated
#       like a Task instance
task_group = manager.pending_tasks

# Tasks are able to be configured directly after obtaining them.
# Get tasks and add a tag 'tag-I' to them
manager.pending_tasks.add_tag("tag-I")
```

3.2.2 Task Scheduling

Job is able to be scheduled as periodic, daily, weekly, monthly or even non-periodic *Task*. It depends on what you need.

Every *Task* should be scheduled before activating them.
It will not be able to activate the task activity if the task has not been scheduled.

Periodic Task

Periodic task will do the job every time interval you set.

Please note that the periodic task will *do the job once immediately* when *starting* the task.

Use *period* method to schedule a periodic task.

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",))

# Periodic Task
# Set time interval to 300 seconds
task.period(300)
```

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",))

# Periodic Task
# Set time interval to 1 hour and 30 minutes
# String format should match `HH:MM:SS`
task.period("01:30:00")
```

```
from schedule_manager import Task
from datetime import timedelta
```

(continues on next page)

(continued from previous page)

```
task = Task(name="task", job=print, args=("Hello task",))

# Periodic Task
# Set time interval to 20 minutes
task.period(timedelta(minutes=20))
```

Daily Task

Daily task will do the job at particular time everyday.

Use *period_day_at* method to schedule a daily task.

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",))

# Daily Task
# Job will be done at 15:00 everyday
# String format should match `HH:MM:SS`
task.period_day_at("15:00:00")
```

Weekly Task

Weekly task will do the job at particular time every week.

Use *period_week_at* method to schedule a weekly task.

Argument week_day can be one of following value:

- *Monday*
- *Tuesday*
- *Wednesday*
- *Thursday*
- *Friday*
- *Saturday*
- *Sunday*

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",))

# Weekly Task
# Job will be done at 15:00 on Tuesday every week
task.period_week_at("15:00:00", week_day="Tuesday")
```

Monthly Task

Monthly task will do the job at particular time every month. Task will skip job activity that month if specific date is not available.

Use *period_month_at* method to schedule a monthly task.

Argument **day** should be in 1 ~ 31.

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",))

# Monthly Task
# Job will be done at 12:00 on day 10 every month
task.period_month_at("12:00:00", day=10)


# Job will be done only in 1/31, 3/31, 5/31, 7/31, 8/31, 10/31
# and 12/31 every year
task.period_month_at("12:00:00", day=31)
```

Non-periodic Task

If you want to schedule a task to do job in a specific times, non-periodic task is appropriate for you.

Use *nonperiodic* method to schedule task as a non-periodic task after assigning time interval between job activity to the task.

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",))

# Non-periodic Task
# Job will be done 10 times.
task.period(60).nonperiodic(10)
```

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",))

# Schedule a 5 days daily task
task.period_day_at("15:00:00").nonperiodic(5)
```

3.2.3 Postpone the Start of a Task

Sometimes we do not want to start the task immediately, we need to start it later. So *Task* object provides two ways for us to postpone the start time of the task.

It is convenient especially for scheduling a periodic task.

Delay Task Start Time

We can set a delay time to postpone the start time of the task.

Use *delay* method to configure a delay time to a task.

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",))

```

(continues on next page)

(continued from previous page)

```
# Task will be started after 600 seconds
task.period(60).delay(600)
```

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",))

# Task will be started after 1 hour
# String format should match `HH:MM:SS`
task.period(60).delay("01:00:00")
```

```
from schedule_manager import Task
from datetime import timedelta

task = Task(name="task", job=print, args=("Hello task",))

# Task will be started after 50 minutes
task.period(60).delay(timedelta(minutes=50))
```

Start Task at Particular Time

We can set a start time to tell the task when the task should be started.

Use *start_at* method to configure a start time to a task.

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",))

# Task will be started at 21:00
# String format should match `HH:MM:SS`
task.period(60).start_at("21:00:00")
```

```
from schedule_manager import Task
from datetime import datetime

target = datetime(year=2020, month=8, day=15,
                  hour=12, minute=30, second=0)


task = Task(name="task", job=print, args=("Hello task",))

# Task will be started at 2020-08-15 12:30:00
task.period(60).start_at(target)
```

3.2.4 Activate Task

After we have configured a *Task*, we need to activate the task to let it do the job on schedule.

After activating the task, we are able to *stop* or *pause* the task in any time.

Start the Task

We can use *start* method to activate the task when we have configured already it.

Activation is *not allowed* if the task has not been scheduled. We need to *schedule the task* first.

Please note that the task *is not able to be configured* any more after we activate the task.

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",))

# Schedule the task first.
task.period(300)

# Activate the task
task.start()
```

Stop the Task

We can use *stop* method to stop the task's activity if we do not need it anymore.

The *stop* method is *not allowed* if the task is not activated.

Because *Task* class is inherited from `Thread` class, the task is not able to be activated again. (See here for more information.)

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",))
task.period(300).start()

# Stop the task
task.stop()
```

If the task is registered in a *ScheduleManager*, the *stop* method will unregister the task from the schedule manager automatically.

```
>>> from schedule_manager import ScheduleManager
>>> from schedule_manager import Task
>>> manager = ScheduleManager()
>>> task = Task(name="task", job=print, args=("Hello task",))
>>> manager.register(task).period_day_at("15:00:00").start()
>>> "task" in manager
True
>>> task.stop()
>>> "task" in manager
False
>>> task.start()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "D:\Application\Python3\Lib\site-packages\schedule_manager\manager.py", line␣
↪995, in start
    super().start()
  File "D:\Application\Python3\Lib\threading.py", line 848, in start
    raise RuntimeError("threads can only be started once")
RuntimeError: threads can only be started once
```

Pause the Task

We can use *pause* method to pause the task's activity if we want to reconfigure the task or want to stop the task for a while.

The *pause* method is *not allowed* if the task is not activated.

Please note that *pause* method is *only allowed* when the task is registerd in a *ScheduleManager* object.

```
from schedule_manager import Task

task = Task(name="task", job=print, args=("Hello task",))
task.period(300).start()

# Pause the task
task.pause()
```

```
>>> from schedule_manager import ScheduleManager
>>> from schedule_manager import Task
>>> manager = ScheduleManager()
>>> task = Task(name="task", job=print, args=("Hello task",))
>>> manager.register(task).period_day_at("15:00:00").start()
>>> "task" in manager
True
>>> task.pause()
>>> "task" in manager
True
>>> manager.task("task").start()
>>> manager.task("task").stop()
```

Check out *Advance Usage* section for more information.

3.3 Advanced Usage

This part of documentation will introduce extra usage of Schedule Manager.

3.3.1 Task Information

We are able to get task information from *Task* object.

Check Activation

Task provides *is_running* property to check the activation of the task.

```
>>> from schedule_manager import Task
>>> task = Task(name="task", job=print, args=("Hello task",))
>>> task.is_running
False
>>> task.period_day_at("15:00:00").start()
>>> task.is_running
True
```

Job Next Run Time

Task provides *next_run* property to check when the job will be done next time .

```
>>> from schedule_manager import Task
>>> from datetime import datetime
>>> task = Task(name="task", job=print, args=("Hello task",))
>>> datetime.now()
datetime.datetime(2020, 8, 9, 12, 14, 10, 361827)
>>> task.period_day_at("15:00:00").start()
>>> task.next_run
datetime.datetime(2020, 8, 9, 15, 0, 0, 802553)
```

3.3.2 Task Behavior with *ignore_skipped* flag

The `ignore_skipped` flag is used to control the behavior if the job take *more* time than time interval configured to the task.

If `ignore_skipped` flag is set to *True*, *Task* will ignore the overdue work.

Let's use examples for explanation.

Set *ignore_skipped* flag to *True*

We will use following python code for testing.

```
"""test_ignore_skipped_True.py"""

from schedule_manager import Task
import time

# A task will take 5 seconds to do the job
task = Task(job=time.sleep, args=(5,), ignore_skipped=True)

# Set time interval to 2 second
task.period(2)

task.start()

while True:
    # Show job next run time
    print(task.next_run.strftime("%H:%M:%S"))

    time.sleep(1)
```

We can see that the works at following time are skipped when *ignore_skipped* flag is set to *True*.

- *19:34:00*
- *19:34:02*
- *19:34:06*
- *19:34:08*
- *19:34:12*
- *19:34:14*

Because the task is busy at those time.

```
$ python test_ignore_skipped_True.py
19:33:58
19:33:58
19:33:58
19:33:58
19:33:58
19:33:58
19:34:04
19:34:04
19:34:04
19:34:04
19:34:04
19:34:04
19:34:10
19:34:10
19:34:10
19:34:10
19:34:10
19:34:10
19:34:16
19:34:16
19:34:16
```

Set *ignore_skipped* flag to *False*

Now we use following python code for another test.

```
"""test_ignore_skipped_False.py"""

from schedule_manager import Task
import time

# A task will take 5 seconds to do the job
task = Task(job=time.sleep, args=(5,), ignore_skipped=False)

# Set time interval to 2 second
task.period(2)

task.start()

while True:
    # Show job next run time
    print(task.next_run.strftime("%H:%M:%S"))

    time.sleep(1)
```

If *ignore_skipped* flag is set to *False*, We can see that all works are not skipped even those works are overdue works.

```
$ python test_ignore_skipped_False.py
19:37:39
19:37:39
19:37:39
19:37:39
19:37:39
```

(continues on next page)

(continued from previous page)

```
19:37:39
19:37:41
19:37:41
19:37:41
19:37:41
19:37:41
19:37:41
19:37:43
19:37:43
19:37:43
19:37:43
19:37:43
19:37:45
19:37:45
```

3.3.3 Task Count

ScheduleManager provides *count* property to show how many tasks are registered in this manager.

```
>>> from schedule_manager import ScheduleManager
>>> manager = ScheduleManager()
>>> manager.register_task(job=print, args=("Task 1",))
Task<(Task-47844607c2ef4354903824cf1abc70be, initial daemon, None)>
>>> manager.register_task(job=print, args=("Task 2",))
Task<(Task-d512b84c33384a6b9bcb09c5f5f00207, initial daemon, None)>
>>> manager.register_task(job=print, args=("Task 3",))
Task<(Task-0cdeb2892d244e1fb489ab943a5a70af, initial daemon, None)>
>>> manager.count    # how many tasks are registered in this manager
3
```

TaskGroup also provides *count* property to show how many tasks we keep in the task group.

```
>>> from schedule_manager import ScheduleManager
>>> manager = ScheduleManager()
>>> task1 = manager.register_task(job=print, args=("Task 1",))
>>> task2 = manager.register_task(job=print, args=("Task 2",))
>>> task3 = manager.register_task(job=print, args=("Task 3",))
>>> task1.period_day_at("22:00:00").start()
>>> manager.pending_tasks.count
2
```

CHAPTER 4

API Documentation

4.1 Schedule Manager API Guide

This part of documentation provides all the interfaces of Schedule Manager. Detailed information about a specific function, class or method can be found here.

4.1.1 ScheduleManager Object

class schedule_manager.**ScheduleManager**

Task schedule manager.

count

Number of tasks registered in the schedule manager.

Type int

all_tasks

Get all tasks.

Type *TaskGroup*

running_tasks

Get all running tasks.

Type *TaskGroup*

pending_tasks

Get all pending tasks.

Type *TaskGroup*

task(*name*)

Get task registerd in schedule manager by name.

Parameters **name** (*str*) – Task name.

Returns Task instance.

Return type *Task*

Raises `TaskNotFoundError` – Task is not registered in schedule manager.

tasks(*tag*)
Get tasks registerd in schedule manager by name.

Parameters **tag** (*Union[obj, list]*) – Tag or tag list.

Returns TaskGroup instance.

Return type *TaskGroup*

register(*task*)
Register a task.

Parameters **task** (`Task`) – Task.

Returns Registered task instance.

Return type *Task*

Raises `TaskNameDuplicateError` – Duplicate task name.

register_task(*job*, *name=None*, *args=()*, *kwargs=None*, *ignore_skipped=True*, *daemon=True*)
Create and register a task.

Parameters

- **job** (*callable*) – Job to be scheduled.
- **name** (*str*) – Task name. By default, a unique name is constructed.
- **args** (*tuple*) – Argument tuple for the job invocation. Defaults to ().
- **kwargs** (*dict*) – Dictionary of keyword arguments for the job invocation. Defaults to {}.
- **ignore_skipped** (*bool*) – Set True to ignore skipped job if time spent on job is longer than the task cycle time. Defaults to True.
- **daemon** (*bool*) – Set True to use as a daemon task. Defaults to True.

Returns Registered task instance.

Return type *Task*

Raises `TaskNameDuplicateError` – Duplicate task name.

unregister(*name=None*, *tag=None*)
Unregister the task.

Parameters

- **name** (*str*) – Unregister task by name.
- **tag** (*Union[obj, list]*) – Unregister tasks by tag or by a list of tags.

4.1.2 Task Object

class schedule_manager.**Task**(*job*, *name=None*, *args=()*, *kwargs=None*, *ignore_skipped=True*, *daemon=True*)
Thread-based Task.

Task will be considered as periodic task by default.

Task is able to registered in *ScheduleManager* or run directly.

Parameters

- **job** (*callable*) – Job to be scheduled as a task.
- **name** (*str*) – Task name. By default, a unique name is constructed.
- **args** (*tuple*) – Argument tuple for the job invocation. Defaults to ().
- **kwargs** (*dict*) – Dictionary of keyword arguments for the job invocation. Defaults to {}.
- **ignore_skipped** (*bool*) – Set True to ignore skipped job if time spent on job is longer than the task cycle time. Defaults to True.
- **daemon** (*bool*) – Set True to use as a daemon task. Defaults to True.

name
Task name.

Type str

daemon
A boolean value indicating whether this task is based on a daemon thread.

See for threading.Thread.daemon more detail.

Type bool

next_run
Datetime when the job run at next time.

Type datetime

is_running
Return True if the task is running.

Type bool

manager
Schedule manager which manages current task.

Type *ScheduleManager*

tag
Tag list of the task.

Type list

add_tag(*tag*)
Add tag to task.

Parameters **tag** (*obj*) – Tag.

Returns Invoked task instance.

Return type *Task*

add_tags(*tags*)
Add a list of tags to task.

Parameters **tags** (*iterable*) – Tag list.

Returns Invoked task instance.

Return type *Task*

remove_tag(*tag*)
Remove tag from task.

Parameters **tag** (*obj*) – Tag.

Returns Invoked task instance.

Return type *Task*

remove_tags(*tags*)
Remove a list of tags from task.

Parameters **tags** (*iterable*) – Tag list.

Returns Invoked task instance.

Return type *Task*

set_tags(*tags*)
Set tag list to task.

Replace old tag list.

Parameters **tags** (*iterable*) – Tag list.

Returns Invoked task instance.

Return type *Task*

delay(*interval=None*)
Delay task start time.

Parameters **interval** (*Union[str, timedelta, int]*) – Time interval. A string with format *HH:MM:SS* or `timedelta` or int in seconds. Or set None to cancel task delay time. Defaults to None.

Returns Invoked task instance.

Return type *Task*

Raises `TimeFormatError` – Invalid time format.

start_at(*at_time=None*)
Set task start time.

Specify a particular time that the job should be start.

Parameters **at_time** (*Union[str, datetime]*) – Start time. A string or `datetime`. A string can be in one of the following formats: [*HH:MM:SS*, *mm-dd HH:MM:SS*]. Or set None to cancel task start time. Defaults to None.

Returns Invoked task instance.

Return type *Task*

Raises `TimeFormatError` – Invalid time format.

nonperiodic(*count*)
See as an non-periodic task.

Parameters **count** (*int*) – Do the job for a certain number of times.

Returns Invoked task instance.

Return type *Task*

periodic()
See as an periodic task.

Returns Invoked task instance.

Return type *Task*

period(*interval*)
Scheduling periodic task.

Parameters **interval** (*Union[str, timedelta, int]*) – Time interval. A string with format *HH:MM:SS* or `timedelta` or int in seconds.

Returns Invoked task instance.

Return type *Task*

Raises `TimeFormatError` – Invalid time format.

period_at(*unit='day'*, *at_time='00:00:00'*, *week_day='Monday'*, *day=1*)
Scheduling periodic task.

Specify a particular time that the job should be run at.

Parameters

- **unit** (*str*) – Time unit of the periodic task. Defaults to *day*. The following unit is available: 1. *day*: Run job everyday. 2. *week*: Run job every week. 3. *month*: Run job every month.
- **at_time** (*str*) – Time to do the job. A string with format *HH:MM:SS*. Defaults to *00:00:00*.
- **week_day** (*str*) – Week to do the job. Defaults to *Monday*. This argument will only be used is unit is *week*. A string should be one of following value: [*"Monday"*, *"Tuesday"*, *"Wednesday"*, *"Thursday"*, *"Friday"*, *"Saturday"*, *"Sunday"*]
- **day** (*int*) – Day to do the job. Defaults to 1. This argument will only be used is unit is *month*. Value should be in 1 ~ 31. Job will be skipped if specific date is not available.

Returns Invoked task instance.

Return type *Task*

Raises `TimeFormatError` – Invalid time format.

period_day_at(*at_time='00:00:00'*)
Scheduling periodic task.

Specify a particular time that the job should be run at. Job runs everyday.

Parameters

- **at_time** (*str*) – Time to do the job. A string with format *HH:MM:SS*. Defaults to *00:00:00*.
- **week_day** (*str*) – Week to do the job.

Returns Invoked task instance.

Return type *Task*

Raises `TimeFormatError` – Invalid time format.

period_week_at(*at_time='00:00:00'*, *week_day='Monday'*)
Scheduling periodic task.

Specify a particular time that the job should be run at. Job runs every week.

Parameters

- **at_time** (*str*) – Time to do the job. A string with format *HH:MM:SS*. Defaults to *00:00:00*.
- **week_day** (*str*) – Week to do the job. Defaults to *Monday*. A string should be one of following value: [*"Monday"*, *"Tuesday"*, *"Wednesday"*, *"Thursday"*, *"Friday"*, *"Saturday"*, *"Sunday"*]

Returns Invoked task instance.

Return type *Task*

Raises TimeFormatError – Invalid time format.

period_month_at(*at_time='00:00:00'*, *day=1*)
Scheduling periodic task.

Specify a particular time that the job should be run at. Job runs every month.

Parameters

- **at_time** (*str*) – Time to do the job. A string with format *HH:MM:SS*. Defaults to *00:00:00*.
- **day** (*int*) – Day to do the job. Defaults to 1. Value should be in 1 ~ 31. Job will be skipped if specific date is not available.

Returns Invoked task instance.

Return type *Task*

Raises TimeFormatError – Invalid time format.

start()
Start the Task's activity.

stop()
Stop the Task's activity.

pause()
Pause the Task's activity.

Works only the task is registered into *ScheduleManager*.

4.1.3 TaskGroup Object

class schedule_manager.**TaskGroup**(*tasks=None*)
Task group.

A set of tasks.

count
Number of tasks contained in the group.

Type int

set_manager(*manager=None*)
Change schedule manager of all tasks.

Task will be unregistered from old manager if it has been registered in a manager.

Parameters **manager** (ScheduleManager) – A exist schedule manager object. Set None to create new schedule manager.

Returns Invoked ScheduleManager instance.

Return type *ScheduleManager*

Raises `TaskNameDuplicateError` – There is a duplicate task name.

add_tag(*tag*)
Add tag to tasks.

Parameters **tag** (*obj*) – Tag.

Returns Invoked TaskGroup instance.

Return type *TaskGroup*

add_tags(*tags*)
Add a list of tags to tasks.

Parameters **tags** (*iterable*) – Tag list.

Returns Invoked TaskGroup instance.

Return type *TaskGroup*

remove_tag(*tag*)
Remove tag from tasks.

Parameters **tag** (*obj*) – Tag.

Returns Invoked TaskGroup instance.

Return type *TaskGroup*

remove_tags(*tags*)
Remove a list of tags from tasks.

Parameters **tags** (*iterable*) – Tag list.

Returns Invoked TaskGroup instance.

Return type *TaskGroup*

set_tags(*tags*)
Set tag list to tasks.

Replace old tag list.

Parameters **tags** (*iterable*) – Tag list.

Returns Invoked TaskGroup instance.

Return type *TaskGroup*

delay(*interval=None*)
Delay task start time.

Parameters **interval** (*Union[str, timedelta, int]*) – Time interval. A string with format *HH:MM:SS* or `timedelta` or int in seconds. Or set None to cancel task delay time. Defaults to None.

Returns Invoked TaskGroup instance.

Return type *TaskGroup*

Raises `TimeFormatError` – Invalid time format.

start_at(*at_time*)
Set task start time.

Specify a particular time that the job should be start.

Parameters **at_time** (*Union[str, datetime]*) – Start time. A string or `datetime`. A string can be in one of the following formats: [*HH:MM:SS*, *mm-dd HH:MM:SS*]. Or set None to cancel task start time. Defaults to None.

Returns Invoked TaskGroup instance.

Return type *TaskGroup*

Raises `TimeFormatError` – Invalid time format.

nonperiodic(*count*)
See as non-periodic tasks.

Parameters **count** (*int*) – Do the job for a certain number of times.

Returns Invoked TaskGroup instance.

Return type *TaskGroup*

periodic()
See as periodic tasks.

Returns Invoked TaskGroup instance.

Return type *TaskGroup*

period(*interval*)
Scheduling periodic tasks.

Parameters **interval** (*Union[str, timedelta, int]*) – Time interval. A string with format *HH:MM:SS* or `timedelta` or int in seconds.

Returns Invoked TaskGroup instance.

Return type *TaskGroup*

Raises `TimeFormatError` – Invalid time format.

period_at(*unit='day'*, *at_time='00:00:00'*, *week_day='Monday'*, *day=1*)
Scheduling periodic tasks.

Specify a particular time that the job should be run at.

Parameters

- **unit** (*str*) – Time unit of the periodic task. Defaults to *day*. The following unit is available: 1. *day*: Run job everyday. 2. *week*: Run job every week. 3. *month*: Run job every month.
- **at_time** (*str*) – Time to do the job. A string with format *HH:MM:SS*. Defaults to *00:00:00*.
- **week_day** (*str*) – Week to do the job. Defaults to *Monday*. This argument will only be used is unit is *week*. A string should be one of following value: [*"Monday"*, *"Tuesday"*, *"Wednesday"*, *"Thursday"*, *"Friday"*, *"Saturday"*, *"Sunday"*]
- **day** (*int*) – Day to do the job. Defaults to 1. This argument will only be used is unit is *month*. Value should be in 1 ~ 31. Job will be skipped if specific date is not available.

Returns Invoked TaskGroup instance.

Return type *TaskGroup*

Raises `TimeFormatError` – Invalid time format.

period_day_at(*at_time='00:00:00'*)
Scheduling periodic tasks.

Specify a particular time that the job should be run at. Job runs everyday.

Parameters

- **at_time** (*str*) – Time to do the job. A string with format *HH:MM:SS*. Defaults to *00:00:00*.
- **week_day** (*str*) – Week to do the job.

Returns Invoked TaskGroup instance.

Return type *TaskGroup*

Raises TimeFormatError – Invalid time format.

period_week_at(*at_time='00:00:00'*, *week_day='Monday'*)
Scheduling periodic tasks.

Specify a particular time that the job should be run at. Job runs every week.

Parameters

- **at_time** (*str*) – Time to do the job. A string with format *HH:MM:SS*. Defaults to *00:00:00*.
- **week_day** (*str*) – Week to do the job. Defaults to *Monday*. A string should be one of following value: [*"Monday"*, *"Tuesday"*, *"Wednesday"*, *"Thursday"*, *"Friday"*, *"Saturday"*, *"Sunday"*]

Returns Invoked TaskGroup instance.

Return type *TaskGroup*

Raises TimeFormatError – Invalid time format.

period_month_at(*at_time='00:00:00'*, *day=1*)
Scheduling periodic tasks.

Specify a particular time that the job should be run at. Job runs every month.

Parameters

- **at_time** (*str*) – Time to do the job. A string with format *HH:MM:SS*. Defaults to *00:00:00*.
- **day** (*int*) – Day to do the job. Defaults to 1. Value should be in 1 ~ 31. Job will be skipped if specific date is not available.

Returns Invoked TaskGroup instance.

Return type *TaskGroup*

Raises TimeFormatError – Invalid time format.

start()
Start the Tasks' activity.

stop()
Stop the Tasks' activity.

pause()
Pause the Tasks' activity.

Works only the task is registered into *ScheduleManager*.

4.1.4 Exceptions

class schedule_manager.exceptions.**OperationFailError**
Operation fail exception.

class schedule_manager.exceptions.**TaskNameDuplicateError**
Duplicate task name exception.

class schedule_manager.exceptions.**TaskNotFoundError**
Task is not registered in schedule manager.

class schedule_manager.exceptions.**TimeFormatError**
Time format error.

CHAPTER 5

License

MIT license.

See LICENSE for for more information.

Python Module Index

S

Index

S

T

U