

## Note:

You are not reading the most recent version of this documentation. [v5.4.0](#) is the latest version available.

This document describes the current stable version of Celery (5.3). For development docs, [go here](#).

# Periodic Tasks

- [Introduction](#)
- [Time Zones](#)
- [Entries](#)
  - [Available Fields](#)
- [Crontab schedules](#)
- [Solar schedules](#)
- [Starting the Scheduler](#)
  - [Using custom scheduler classes](#)

## Introduction

**celery beat** is a scheduler; It kicks off tasks at regular intervals, that are then executed by available worker nodes in the cluster.

By default the entries are taken from the [beat\\_schedule](#) setting, but custom stores can also be used, like storing the entries in a SQL database.

You have to ensure only a single scheduler is running for a schedule at a time, otherwise you'd end up with duplicate tasks. Using a centralized approach means the schedule doesn't have to be synchronized, and the service can operate without using locks.

## Time Zones

The periodic task schedules uses the UTC time zone by default, but you can change the time zone used using the [timezone](#) setting.

An example time zone could be *Europe/London*:

```
timezone = 'Europe/London'
```

This setting must be added to your app, either by configuring it directly using (`app.conf.timezone = 'Europe/London'`), or by adding it to your configuration module if you have set one up using `app.config_from_object`. See [Configuration](#) for more information about configuration options.

The default scheduler (storing the schedule in the `celerybeat-schedule` file) will automatically detect that the time zone has changed, and so will reset the schedule itself, but other schedulers may not be so smart (e.g., the Django database scheduler, see below) and in that case you'll have to reset the schedule manually.

 [v: v5.3.6](#) ▼

Celery recommends and is compatible with the new `USE_TZ` setting introduced in Django 1.4.

For Django users the time zone specified in the `TIME_ZONE` setting will be used, or you can specify a custom time zone for Celery alone by using the `timezone` setting.

The database scheduler won't reset when timezone related settings change, so you must do this manually:

```
$ python manage.py shell
>>> from djscheduler.models import PeriodicTask
>>> PeriodicTask.objects.update(last_run_at=None)
```

Django-Celery only supports Celery 4.0 and below, for Celery 4.0 and above, do as follow:

```
$ python manage.py shell
>>> from django_celery_beat.models import PeriodicTask
>>> PeriodicTask.objects.update(last_run_at=None)
```

## Entries

To call a task periodically you have to add an entry to the beat schedule list.

```
from celery import Celery
from celery.schedules import crontab

app = Celery()

@app.on_after_configure.connect
def setup_periodic_tasks(sender, **kwargs):
    # Calls test('hello') every 10 seconds.
    sender.add_periodic_task(10.0, test.s('hello'), name='add every 10')

    # Calls test('hello') every 30 seconds.
    # It uses the same signature of previous task, an explicit name is
    # defined to avoid this task replacing the previous one defined.
    sender.add_periodic_task(30.0, test.s('hello'), name='add every 30')

    # Calls test('world') every 30 seconds
    sender.add_periodic_task(30.0, test.s('world'), expires=10)

    # Executes every Monday morning at 7:30 a.m.
    sender.add_periodic_task(
        crontab(hour=7, minute=30, day_of_week=1),
        test.s('Happy Mondays!'),
    )

@app.task
def test(arg):
    print(arg)

@app.task
def add(x, y):
    z = x + y
    print(z)
```

 v: v5.3.6 ▼

6/21/24, 1:39 AM Periodic Tasks — Celery 5.3.6 documentation

Setting these up from within the `on_after_configure` handler means that we'll not evaluate the app at module level when using `test.s()`. Note that `on_after_configure` is sent after the app is set up, so tasks outside the module where the app is declared (e.g. in a `tasks.py` file located by `celery.Celery.autodiscover_tasks()`) must use a later signal, such as `on_after_finalize`.

The `add_periodic_task()` function will add the entry to the `beat_schedule` setting behind the scenes, and the same setting can also be used to set up periodic tasks manually:

Example: Run the `tasks.add` task every 30 seconds.

```
app.conf.beat_schedule = {
    'add-every-30-seconds': {
        'task': 'tasks.add',
        'schedule': 30.0,
        'args': (16, 16)
    },
}
app.conf.timezone = 'UTC'
```

## Note:

If you're wondering where these settings should go then please see [Configuration](#). You can either set these options on your app directly or you can keep a separate module for configuration.

If you want to use a single item tuple for `args`, don't forget that the constructor is a comma, and not a pair of parentheses.

Using a `timedelta` for the schedule means the task will be sent in 30 second intervals (the first task will be sent 30 seconds after `celery beat` starts, and then every 30 seconds after the last run).

A Crontab like schedule also exists, see the section on [Crontab schedules](#).

Like with `cron`, the tasks may overlap if the first task doesn't complete before the next. If that's a concern you should use a locking strategy to ensure only one instance can run at a time (see for example [Ensuring a task is only executed one at a time](#)).

## Available Fields ¶

- `task`

The name of the task to execute.

Task names are described in the [Names](#) section of the User Guide. Note that this is not the import path of the task, even though the default naming pattern is built like it is.

- `schedule`

The frequency of execution.

This can be the number of seconds as an integer, a `timedelta`, or a `crontab`. You can also define your own custom schedule types, by extending the interface of `schedule`.

- `args`

Positional arguments ([list](#) or [tuple](#)).

 v: v5.3.6 ▼

- *kwargs*

Keyword arguments (**dict**).

- *options*

Execution options (**dict**).

This can be any argument supported by **apply\_async()** – *exchange, routing\_key, expires*, and so on.

- *relative*

If *relative* is true **timedelta** schedules are scheduled “by the clock.” This means the frequency is rounded to the nearest second, minute, hour or day depending on the period of the **timedelta**.

By default *relative* is false, the frequency isn’t rounded and will be relative to the time when **celery beat** was started.

## Crontab schedules

If you want more control over when the task is executed, for example, a particular time of day or day of the week, you can use the **crontab** schedule type:

```
from celery.schedules import crontab

app.conf.beat_schedule = {
    # Executes every Monday morning at 7:30 a.m.
    'add-every-monday-morning': {
        'task': 'tasks.add',
        'schedule': crontab(hour=7, minute=30, day_of_week=1),
        'args': (16, 16),
    },
}
```

The syntax of these Crontab expressions are very flexible.

Some examples:

Example	Meaning
<code>crontab()</code>	Execute every minute.
<code>crontab(minute=0, hour=0)</code>	Execute daily at midnight.
<code>crontab(minute=0, hour='*/3')</code>	Execute every three hours: midnight, 3am, 6am, 9am, noon, 3pm, 6pm, 9pm.
<code>crontab(minute=0, hour='0,3,6,9,12,15,18,21')</code>	Same as previous.
<code>crontab(minute='*/15')</code>	Execute every 15 minutes.
<code>crontab(day_of_week='sunday')</code>	Execute every minute (!) at Sundays.
<code>crontab(minute='*', hour='*', day_of_week='sun')</code>	Same as previous.

 v: v5.3.6 ▾

<code>crontab(minute='*/10', hour='3,17,22', day_of_week='thu,fri')</code>	Execute tasks on Celery 5.3 documentation between 3-4 am, 5-6 pm, and 10-11 pm on Thursdays or Fridays.
<code>crontab(minute=0, hour='*/2,*/3')</code>	Execute every even hour, and every hour divisible by three. This means: at every hour <i>except</i> : 1am, 5am, 7am, 11am, 1pm, 5pm, 7pm, 11pm
<code>crontab(minute=0, hour='*/5')</code>	Execute hour divisible by 5. This means that it is triggered at 3pm, not 5pm (since 3pm equals the 24-hour clock value of "15", which is divisible by 5).
<code>crontab(minute=0, hour='*/3,8-17')</code>	Execute every hour divisible by 3, and every hour during office hours (8am-5pm).
<code>crontab(0, 0, day_of_month='2')</code>	Execute on the second day of every month.
<code>crontab(0, 0, day_of_month='2-30/2')</code>	Execute on every even numbered day.
<code>crontab(0, 0, day_of_month='1-7,15-21')</code>	Execute on the first and third weeks of the month.
<code>crontab(0, 0, day_of_month='11', month_of_year='5')</code>	Execute on the eleventh of May every year.
<code>crontab(0, 0, month_of_year='*/3')</code>	Execute every day on the first month of every quarter.

See [celery.schedules.crontab](#) for more documentation.

## Solar schedules

If you have a task that should be executed according to sunrise, sunset, dawn or dusk, you can use the [solar](#) schedule type:

```
from celery.schedules import solar

app.conf.beat_schedule = {
    # Executes at sunset in Melbourne
    'add-at-melbourne-sunset': {
        'task': 'tasks.add',
        'schedule': solar('sunset', -37.81753, 144.96715),
        'args': (16, 16),
    },
}
```

The arguments are simply: `solar(event, latitude, longitude)`

Be sure to use the correct sign for latitude and longitude:

Sign	Argument	Meaning
+	latitude	North
-	latitude	South
+	longitude	East

Possible event types are:

Event	Meaning
dawn_astronomical	Execute at the moment after which the sky is no longer completely dark. This is when the sun is 18 degrees below the horizon.
dawn_nautical	Execute when there's enough sunlight for the horizon and some objects to be distinguishable; formally, when the sun is 12 degrees below the horizon.
dawn_civil	Execute when there's enough light for objects to be distinguishable so that outdoor activities can commence; formally, when the Sun is 6 degrees below the horizon.
sunrise	Execute when the upper edge of the sun appears over the eastern horizon in the morning.
solar_noon	Execute when the sun is highest above the horizon on that day.
sunset	Execute when the trailing edge of the sun disappears over the western horizon in the evening.
dusk_civil	Execute at the end of civil twilight, when objects are still distinguishable and some stars and planets are visible. Formally, when the sun is 6 degrees below the horizon.
dusk_nautical	Execute when the sun is 12 degrees below the horizon. Objects are no longer distinguishable, and the horizon is no longer visible to the naked eye.
dusk_astronomical	Execute at the moment after which the sky becomes completely dark; formally, when the sun is 18 degrees below the horizon.

All solar events are calculated using UTC, and are therefore unaffected by your timezone setting.

In polar regions, the sun may not rise or set every day. The scheduler is able to handle these cases (i.e., a sunrise event won't run on a day when the sun doesn't rise). The one exception is `solar_noon`, which is formally defined as the moment the sun transits the celestial meridian, and will occur every day even if the sun is below the horizon.


Twilight is defined as the period between dawn and sunrise; and between sunset and dusk. You can schedule an event according to "twilight" depending on your definition of twilight (civil, nautical, or astronomical), and whether you want the event to take place at the beginning or end of twilight, using the appropriate event from the list above.

See [celery.schedules.solar](#) for more documentation.

## Starting the Scheduler

To start the **celery beat** service:

```
$ celery -A proj beat
```

You can also embed *beat* inside the worker by enabling the workers **-B** option, this is convenient if you'll never run more than one worker node, but it's not commonly used and for that reason isn't recommended  v: v5.3.6 ▼ for production use:

Beat needs to store the last run times of the tasks in a local database file (named *celerybeat-schedule* by default), so it needs access to write in the current directory, or alternatively you can specify a custom location for this file:

```
$ celery -A proj beat -s /home/celery/var/run/celerybeat-schedule
```

## Note:

To daemonize beat see [Daemonization](#).

## Using custom scheduler classes

Custom scheduler classes can be specified on the command-line (the `--scheduler` argument).

The default scheduler is the `celery.beat.PersistentScheduler`, that simply keeps track of the last run times in a local `shelve` database file.

There's also the `django-celery-beat` extension that stores the schedule in the Django database, and presents a convenient admin interface to manage periodic tasks at runtime.

To install and use this extension:

1. Use **pip** to install the package:

```
$ pip install django-celery-beat
```

2. Add the `django_celery_beat` module to `INSTALLED_APPS` in your Django project's `settings.py`:

```
INSTALLED_APPS = (  
    ...,  
    'django_celery_beat',  
)
```

Note that there is no dash in the module name, only underscores.

3. Apply Django database migrations so that the necessary tables are created:

```
$ python manage.py migrate
```

4. Start the **celery beat** service using the `django_celery_beat.schedulers:DatabaseScheduler` scheduler:

```
$ celery -A proj beat -l INFO --scheduler django_celery_beat.schedulers:DatabaseScheduler
```

Note: You may also add this as the `beat_scheduler` setting directly.

5. Visit the Django-Admin interface to set up some periodic tasks.



**Private docs hosting** for any Docs as Code tool.  
Start a trial with **Read the Docs for Business**.

Ad by EthicalAds · 