CircuitPython uschedule Library Documentation

Release 1.0

Nathan Byrd

Mar 27, 2021

CONTENTS

Reduced version of the schedule library for CircuitPython

This library is a reduced version of the Python schedule library. Credit to Dan Bader (dbader) for this excellent library.

CHAPTER

ONE

DEPENDENCIES

This driver depends on:

- Adafruit CircuitPython
- Adafruit CircuitPython Datetime

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the Adafruit library and driver bundle or individual libraries can be installed using circup.Installing from PyPI
==========

On supported GNU/Linux systems like the Raspberry Pi, you can install the driver locally from PyPI. To install for current user:

```
pip3 install adafruit-circuitpython-uschedule
```

To install system-wide (this may be required in some cases):

```
sudo pip3 install adafruit-circuitpython-uschedule
```

To install in a virtual environment in your current project:

```
mkdir project-name && cd project-name
python3 -m venv .env
source .env/bin/activate
pip3 install adafruit-circuitpython-uschedule
```

CHAPTER

TWO

USAGE EXAMPLE

```
import time
import uschedule as schedule

def greet():
    print("Hello, world!")

""" Note: pass the function name, like greet, not greet():"""
schedule.every(10).seconds.do(greet)

while True:
    schedule.run_pending()
    time.sleep(1)
```

CHAPTER

THREE

CONTRIBUTING

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER

FOUR

DOCUMENTATION

Read the module documentation.

CHAPTER

FIVE

TABLE OF CONTENTS

5.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/uschedule_simpletest.py

```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Nathan Byrd

# SPDX-License-Identifier: MIT

import time
import uschedule as schedule


def greet():
    print("Hello, world!")


# Note: pass functions, not function calls - i.e. "greet", not "greet()"

# schedule every 10 seconds
schedule.every(10).seconds.do(greet)

# schedule every 10 minutes
schedule.every(10).minutes.do(greet)

# schedule once a day
schedule.every().day.at("10:30").do(greet)

# schedule from 5 to 10 minutes
schedule.every(5).to(10).minutes.do(greet)

# schedule on a particular day
schedule.every().monday.do(greet)

# schedule day and time
schedule.every().wednesday.at("13:15").do(greet)

# schedule once a minute at seventeen seconds
schedule.every().minute.at(":17").do(greet)


```

(continues on next page)

(continued from previous page)

```
while True:
    # Run any pending jobs
    schedule.run_pending()
    time.sleep(1)
```

5.2 Additional utilities

Additional utilities that are available

Listing 2: examples/uschedule_util.py

```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Nathan Byrd

# SPDX-License-Identifier: MIT

import time
import uschedule as schedule


def greet():
    print("Hello, world!")


# schedule every 10 seconds
schedule.every(10).seconds.do(greet)

# Get the datetime of next run
next_run_datetime = schedule.next_run()

# Get the number of seconds until the next run
next_run_seconds = schedule.idle_seconds()

# cancel all jobs
schedule.clear()

# schedule every 1 second
schedule.every(1).second.do(greet)

while True:
    # Run any pending jobs
    schedule.run_pending()
    time.sleep(1)
```

5.3 Real Time Clock

This module works great in combination with a Real Time Clock (RTC), if one is available on your device. For example:

Listing 3: examples/uschedule_rtc.py

```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Nathan Byrd

# SPDX-License-Identifier: MIT

import time
import rtc
import busio
import board
import adafruit_pcf8523
import uschedule as schedule


def greet():
    print("Hello, world!")


i2c = busio.I2C(board.SCL, board.SDA)
rtc_device = adafruit_pcf8523.PCF8523(i2c)
rtc.RTC().datetime = rtc_device.datetime

# schedule every 10 seconds
schedule.every(10).seconds.do(greet)

# schedule every 10 minutes
schedule.every(10).minutes.do(greet)

# schedule once a day
schedule.every().day.at("10:30").do(greet)

# schedule from 5 to 10 minutes
schedule.every(5).to(10).minutes.do(greet)

# schedule on a particular day
schedule.every().monday.do(greet)

# schedule day and time
schedule.every().wednesday.at("13:15").do(greet)

# schedule once a minute at seventeen seconds
schedule.every().minute.at(":17").do(greet)


while True:
    # Run any pending jobs
    schedule.run_pending()
    time.sleep(1)
```

Reduced version of the schedule library for CircuitPython

Based on:

5.4 schedule

Python job scheduling for humans.

github.com/dbader/schedule

An in-process scheduler for periodic jobs that uses the builder pattern for configuration. Schedule lets you run Python functions (or any other callable) periodically at pre-determined intervals using a simple, human-friendly syntax.

Inspired by Addam Wiggins' article "Rethinking Cron" [1] and the "clockwork" Ruby module [2][3].

Features:

- A simple to use API for scheduling jobs.
- Very lightweight and no external dependencies.
- Excellent test coverage.
- Tested on Python 3.6, 3.7, 3.8, 3.9

Usage:

```
>>> import uschedule as schedule
>>> import time
```

```
>>> def job():
>>>     print("I'm working on stuff.")
```

```
>>> schedule.every(10).minutes.do(job)
>>> schedule.every(5).to(10).days.do(job)
>>> schedule.every().hour.do(job)
>>> schedule.every().day.at("10:30").do(job)
```

```
>>> while True:
>>>     schedule.run_pending()
>>>     time.sleep(1)
```

[1] https://adam.herokuapp.com/past/2010/4/13/rethinking_cron/ [2] https://github.com/Rykian/clockwork [3] https://adam.herokuapp.com/past/2010/6/30/replace_cron_with_clockwork/

class uschedule.**CancelJob**
Can be returned from a job to unschedule itself.

exception uschedule.**IntervalError**
An improper interval was used

class uschedule.**Job**(*interval: int*, *scheduler: Optional[*uschedule.Scheduler*] = None*)
A periodic job as used by *Scheduler*.

Parameters

- **interval** – A quantity of a certain time unit
- **scheduler** – The *Scheduler* instance that this job will register itself with once it has been fully configured in *Job.do()*.

Every job runs at a given fixed time interval that is defined by:

- a *time unit*
- a quantity of time units defined by interval

A job is usually created and returned by *Scheduler.every()* method, which also defines its interval.

at (*time_str*)
Specify a particular time that the job should be run at.

Parameters **time_str** – A string in one of the following formats:

- For daily jobs -> 'HH:MM:SS' or 'HH:MM'
- For hourly jobs -> 'MM:SS' or ':MM'
- For minute jobs -> ':SS'

The format must make sense given how often the job is repeating; for example, a job that repeats every minute should not be given a string in the form 'HH:MM:SS'. The difference between ':MM' and ':SS' is inferred from the selected time-unit (e.g. every().hour.at(':30') vs. every().minute.at(':30')).

Returns The invoked job instance

property day
Specify the type of an interval as day. Only works if the interval is set to 1

Raises ***IntervalError*** – Thrown if the interval is not 1

Returns Returns self

Return type uschedule

property days
Specify the type of an interval as days.

Returns Returns self

Return type uschedule

do (*job_func*, **args*, ***kwargs*)
Specifies the job_func that should be called every time the job runs.

Any additional arguments are passed on to job_func when the job runs.

NOTE: This version does not support arguments

Parameters **job_func** – The function to be scheduled. This should be a callable function name, not a call. i.e. "greet", not "greet()"

Returns The invoked job instance

property friday
Set the target day of the week as Friday. Only works for weekly jobs.

Raises ***IntervalError*** – Thrown if interval is not weekly

Returns Returns self

Return type uschedule

property hour
Specify the type of an interval as hour. Only works if the interval is set to 1

Raises ***IntervalError*** – Thrown if the interval is not 1

Returns Returns self

Return type uschedule

property hours
Specify the type of an interval as hours.

Returns Returns self

Return type uschedule

property minute
Specify the type of an interval as minute. Only works if the interval is set to 1

Raises *IntervalError* – Thrown if the interval is not 1

Returns Returns self

Return type uschedule

property minutes
Specify the type of an interval as minutes.

Returns Returns self

Return type uschedule

property monday
Set the target day of the week as Monday. Only works for weekly jobs.

Raises *IntervalError* – Thrown if interval is not weekly

Returns Returns self

Return type uschedule

run()
Run the job and immediately reschedule it. If the job's deadline is reached (configured using .until()), the job is not run and CancelJob is returned immediately. If the next scheduled run exceeds the job's deadline, CancelJob is returned after the execution. In this latter case CancelJob takes priority over any other returned value.

Returns The return value returned by the 'job_func', or CancelJob if the job's deadline is reached.

property saturday
Set the target day of the week as Saturday. Only works for weekly jobs.

Raises *IntervalError* – Thrown if interval is not weekly

Returns Returns self

Return type uschedule

property second
Specify the type of an interval as a single second. Only works if the interval is set to 1

Raises *IntervalError* – Thrown if the interval is not 1

Returns Returns self

Return type uschedule

property seconds
Specify the type of an interval as seconds.

Returns Returns self

Return type uschedule

property should_run
return: `True` if the job should be run now.

property sunday
Set the target day of the week as Sunday. Only works for weekly jobs.

Raises ***IntervalError*** – Thrown if interval is not weekly

Returns Returns self

Return type uschedule

tag(**tags*)
Tags the job with one or more unique identifiers.

Tags must be hashable. Duplicate tags are discarded.

Parameters **tags** – A unique list of `Hashable` tags.

Returns The invoked job instance

property thursday
Set the target day of the week as Thursday. Only works for weekly jobs.

Raises ***IntervalError*** – Thrown if interval is not weekly

Returns Returns self

Return type uschedule

to(*latest: int*)
Schedule the job to run at an irregular (randomized) interval.

The job's interval will randomly vary from the value given to 'every' to 'latest'. The range defined is inclusive on both ends. For example, 'every(A).to(B).seconds' executes the job function every N seconds such that A <= N <= B.

Parameters **latest** – Maximum interval between randomized job runs

Returns The invoked job instance

property tuesday
Set the target day of the week as Tuesday. Only works for weekly jobs.

Raises ***IntervalError*** – Thrown if interval is not weekly

Returns Returns self

Return type uschedule

until(*until_time*)
Schedule job to run until the specified moment.

The job is canceled whenever the next run is calculated and it turns out the next run is after the until_time. The job is also canceled right before it runs, if the current time is after until_time. This latter case can happen when the the job was scheduled to run before until_time, but runs after until_time.

If until_time is a moment in the past, ScheduleValueError is thrown.

Parameters **until_time** – A moment in the future representing the latest time a job can be run. If only a time is supplied, the date is set to today. The following formats are accepted:

- datetime.datetime
- datetime.timedelta
- datetime.time

- String in one of the following formats: "%Y-%m-%d %H:%M:%S", "%Y-%m-%d %H:%M", "%Y-%m-%d", "%H:%M:%S", "%H:%M" as defined by strptime() behaviour. If an invalid string format is passed, ScheduleValueError is thrown.

Returns The invoked job instance

property wednesday
Set the target day of the week as Wednesday Only works for weekly jobs.

Raises ***IntervalError*** – Thrown if interval is not weekly

Returns Returns self

Return type uschedule

property week
Specify the type of an interval as week Only works if the interval is set to 1

Raises ***IntervalError*** – Thrown if the interval is not 1

Returns self

Return type uschedule

property weeks
Specify the type of an interval as weeks

Returns Returns self

Return type uschedule

exception uschedule.**ScheduleError**
Base schedule exception

exception uschedule.**ScheduleValueError**
Base schedule value error

class uschedule.**Scheduler**
Objects instantiated by the *Scheduler* are factories to create jobs, keep record of scheduled jobs and handle their execution.

cancel_job(*job:* uschedule.Job) → None
Delete a scheduled job.

Parameters **job** – The job to be unscheduled

clear(*tag=None*) → None
Deletes scheduled jobs marked with the given tag, or all jobs if tag is omitted.

Parameters **tag** – An identifier used to identify a subset of jobs to delete

every(*interval: int = 1*) → *uschedule.Job*
Schedule a new periodic job.

Parameters **interval** – A quantity of a certain time unit

Returns An unconfigured *Job*

get_jobs(*tag=None*)
Gets scheduled jobs marked with the given tag, or all jobs if tag is omitted.

Parameters **tag** – An identifier used to identify a subset of jobs to retrieve

property idle_seconds
return: Number of seconds until *next_run* or None if no jobs are scheduled

property next_run
Datetime when the next job should run.

Returns A `datetime` object or None if no jobs scheduled

run_all(*delay_seconds: int = 0*) → None
Run all jobs regardless if they are scheduled to run or not.

A delay of 'delay' seconds is added between each job. This helps distribute system load generated by the jobs more evenly over time.

Parameters **delay_seconds** – A delay added between every executed job

run_pending() → None
Run all jobs that are scheduled to run.

Please note that it is *intended behavior that run_pending() does not run missed jobs*. For example, if you've registered a job that should run every minute and you only call run_pending() in one hour increments then your job won't be run 60 times in between but only once.

uschedule.**cancel_job**(*job:* uschedule.Job) → None
Calls *cancel_job* on the *default scheduler instance*.

uschedule.**clear**(*tag=None*) → None
Calls *clear* on the *default scheduler instance*.

uschedule.**default_scheduler = <uschedule.Scheduler object>**
Default *Scheduler* object

uschedule.**every**(*interval: int = 1*) → *uschedule.Job*
Calls *every* on the *default scheduler instance*.

uschedule.**get_jobs**(*tag=None*)
Calls *get_jobs* on the *default scheduler instance*.

uschedule.**idle_seconds**()
Calls *idle_seconds* on the *default scheduler instance*.

uschedule.**jobs = []**
Default *Jobs* list

uschedule.**next_run**()
Calls *next_run* on the *default scheduler instance*.

uschedule.**repeat**(*job*, **args*, ***kwargs*)
Decorator to schedule a new periodic job.

Any additional arguments are passed on to the decorated function when the job runs.

Parameters **job** – a *Jobs*

uschedule.**run_all**(*delay_seconds: int = 0*) → None
Calls *run_all* on the *default scheduler instance*.

uschedule.**run_pending**() → None
Calls *run_pending* on the *default scheduler instance*.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

u

INDEX