EAScheduler

spaceman_spiff

Aug 02, 2022

CONTENTS:

CHAPTER

ONE

EASY ASYNC SCHEDULER

1.1 Interaction

1.1.1 SchedulerView

class eascheduler.**SchedulerView**(*scheduler*, *executor*)

at(*time*, *callback*, **args*, ***kwargs*)

Create a a job that will run at a specified time.

Parameters

- **time** (Union[None, datetime, timedelta, time, int, float]) –
- **callback** – Function which will be called
- **args** – Positional arguments that will be passed to the function
- **kwargs** – Keyword arguments that will be passed to the function

Return type
OneTimeJob

Returns
Created job

countdown(*expire_time*, *callback*, **args*, ***kwargs*)

Run a job a specific time after calling `reset()` of the job. Another subsequent call to `reset()` will start the countdown again.

Parameters

- **expire_time** (Union[timedelta, float, int]) – countdown in seconds or a timedelta obj
- **callback** – Function which will be called
- **args** – Positional arguments that will be passed to the function
- **kwargs** – Keyword arguments that will be passed to the function

Return type
CountdownJob

Returns
Created job

every(*start_time*, *interval*, *callback*, **args*, ***kwargs*)

Create a job that will run at a specific interval.

Parameters

- **start_time** (Union[None, datetime, timedelta, time, int, float]) – First execution time
- **interval** (Union[int, float, timedelta]) – Interval how the job is repeated
- **callback** – Function which will be called
- **args** – Positional arguments that will be passed to the function
- **kwargs** – Keyword arguments that will be passed to the function

Return type
ReoccurringJob

Returns
Created job

on_day_of_week(*time*, *weekdays*, *callback*, **args*, ***kwargs*)

Create a job that will run at a certain time on certain days during the week.

Parameters

- **time** (Union[time, datetime]) – Time when the job will run
- **weekdays** (Union[str, Iterable[Union[str, int]]]) – Day group names (e.g. `'all'`, `'weekend'`, `'workdays'`), an iterable with day names (e.g. `['Mon', 'Fri']`) or an iterable with the isoweekday values (e.g. `[1, 5]`).
- **callback** – Function which will be called
- **args** – Positional arguments that will be passed to the function
- **kwargs** – Keyword arguments that will be passed to the function

Return type
DayOfWeekJob

Returns
Created job

on_sun_dawn(*callback*, **args*, ***kwargs*)

Create a job that will run on dawn, requires a location to be set

Parameters

- **callback** – Function which will be called
- **args** – Positional arguments that will be passed to the function
- **kwargs** – Keyword arguments that will be passed to the function

Return type
DawnJob

Returns
Created job

on_sun_dusk(*callback*, **args*, ***kwargs*)

Create a job that will run on dusk, requires a location to be set

Parameters

- **callback** – Function which will be called
- **args** – Positional arguments that will be passed to the function
- **kwargs** – Keyword arguments that will be passed to the function

Return type
DuskJob

Returns
Created job

on_sunrise(*callback*, **args*, ***kwargs*)

Create a job that will run on sunrise, requires a location to be set

Parameters

- **callback** – Function which will be called
- **args** – Positional arguments that will be passed to the function
- **kwargs** – Keyword arguments that will be passed to the function

Return type
SunriseJob

Returns
Created job

on_sunset(*callback*, **args*, ***kwargs*)

Create a job that will run on sunset, requires a location to be set

Parameters

- **callback** – Function which will be called
- **args** – Positional arguments that will be passed to the function
- **kwargs** – Keyword arguments that will be passed to the function

Return type
SunsetJob

Returns
Created job

on_weekends(*time*, *callback*, **args*, ***kwargs*)

Create a job that will run at a certain time on weekends.

Parameters

- **time** (Union[time, datetime]) – Time when the job will run
- **callback** – Function which will be called
- **args** – Positional arguments that will be passed to the function
- **kwargs** – Keyword arguments that will be passed to the function

Return type
DayOfWeekJob

Returns
Created job

on_workdays(*time*, *callback*, **args*, ***kwargs*)

Create a job that will run at a certain time on workdays.

Parameters

- **time** (Union[time, datetime]) – Time when the job will run
- **callback** – Function which will be called
- **args** – Positional arguments that will be passed to the function
- **kwargs** – Keyword arguments that will be passed to the function

Return type
DayOfWeekJob

Returns
Created job

1.2 Jobs

1.2.1 DayOfWeekJob

class eascheduler.jobs.**DayOfWeekJob**(*parent*, *func*)

boundary_func(*func*)

Add a function which will be called when the datetime changes. Use this to implement custom boundaries. Use None to disable the boundary function.

Parameters
func (Optional[Callable[[datetime], datetime]]) – Function which returns a datetime obj, arg is a datetime with the next run time. Return SKIP_EXECUTION together with a reoccurring job to skip the proposed run time.

Return type
DateTimeJobBase

cancel()

Cancel the job.

earliest(*time_obj*)

Set earliest boundary as time of day. None will disable boundary.

Parameters
time_obj (Optional[time]) – time obj, scheduler will not run earlier

Return type
DateTimeJobBase

get_next_run()

Return the next execution timestamp.

Return type
datetime

jitter(*start*, *stop=None*)

Add a random jitter per call in the interval [start <= secs <= stop] to the next run. If stop is omitted start must be positive and the interval will be [-start <= secs <= start] Passing None as start will disable jitter.

Parameters

- **start** (Union[int, float, None]) – Interval start or None to disable jitter
- **stop** (Union[int, float, None]) – Interval stop or None to build interval based on start

Return type
DateTimeJobBase

latest(*time_obj*)

Set latest boundary as time of day. None will disable boundary.

Parameters
time_obj (Optional[time]) – time obj, scheduler will not run later

Return type
DateTimeJobBase

offset(*timedelta_obj*)

Set a constant offset to the calculation of the next run. None will disable the offset.

Parameters
timedelta_obj (Optional[timedelta]) – constant offset

Return type
DateTimeJobBase

remaining()

Returns the remaining time to the next run or None if the job is not scheduled

Return type
Optional[timedelta]

Returns
remaining time as a timedelta or None

time(*time*)

Set a time of day when the job will run.

Parameters
time (Union[time, datetime]) – time

Return type
DayOfWeekJob

weekdays(*weekdays*)

Set the weekdays when the job will run.

Parameters
weekdays (Union[str, Iterable[Union[str, int]]]) – Day group names (e.g. 'all', 'weekend', 'workdays'), an iterable with day names (e.g. ['Mon', 'Fri']) or an iterable with the isoweekday values (e.g. [1, 5]).

Return type
DayOfWeekJob

1.2.2 ExpiringJob

class eascheduler.jobs.**CountdownJob**(*parent*, *func*)

cancel()

Cancel the job.

countdown(*time*)

Set the time after which the job will be executed.

Parameters
time (Union[timedelta, float, int]) – time

Return type
CountdownJob

get_next_run()

Return the next execution timestamp.

Return type
datetime

remaining()

Returns the remaining time to the next run or None if the job is not scheduled

Return type
Optional[timedelta]

Returns
remaining time as a timedelta or None

stop()

Stops the countdown so it can be started again with a call to reset

1.2.3 OneTimeJob

class eascheduler.jobs.**OneTimeJob**(*parent*, *func*)

cancel()

Cancel the job.

get_next_run()

Return the next execution timestamp.

Return type
datetime

remaining()

Returns the remaining time to the next run or None if the job is not scheduled

Return type
Optional[timedelta]

Returns
remaining time as a timedelta or None

1.2.4 ReoccurringJob

class eascheduler.jobs.**ReoccurringJob**(*parent*, *func*)

boundary_func(*func*)

Add a function which will be called when the datetime changes. Use this to implement custom boundaries. Use None to disable the boundary function.

Parameters
func (Optional[Callable[[datetime], datetime]]) – Function which returns a datetime obj, arg is a datetime with the next run time. Return SKIP_EXECUTION together with a reoccurring job to skip the proposed run time.

Return type
DateTimeJobBase

cancel()

Cancel the job.

earliest(*time_obj*)

Set earliest boundary as time of day. None will disable boundary.

Parameters
time_obj (Optional[time]) – time obj, scheduler will not run earlier

Return type
DateTimeJobBase

get_next_run()

Return the next execution timestamp.

Return type
datetime

interval(*interval*)

Set the interval at which the task will run.

Parameters
interval (Union[int, float, timedelta]) – interval in secs or a timedelta obj

Return type
ReoccurringJob

jitter(*start*, *stop=None*)

Add a random jitter per call in the interval [start <= secs <= stop] to the next run. If stop is omitted start must be positive and the interval will be [-start <= secs <= start] Passing None as start will disable jitter.

Parameters

- **start** (Union[int, float, None]) – Interval start or None to disable jitter
- **stop** (Union[int, float, None]) – Interval stop or None to build interval based on start

Return type
DateTimeJobBase

latest(*time_obj*)

Set latest boundary as time of day. None will disable boundary.

Parameters
time_obj (Optional[time]) – time obj, scheduler will not run later

Return type
DateTimeJobBase

`offset`(*timedelta_obj*)

Set a constant offset to the calculation of the next run. `None` will disable the offset.

Parameters
timedelta_obj (Optional[timedelta]) – constant offset

Return type
DateTimeJobBase

`remaining`()

Returns the remaining time to the next run or None if the job is not scheduled

Return type
Optional[timedelta]

Returns
remaining time as a timedelta or None

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

INDEX