crython Documentation

Release 0.2.0

Andrew Hawker

Oct 02, 2020

CONTENTS

Lightweight task scheduler using cron expressions.

CHAPTER ONE

MODULES

1.1 crython/expression

Contains functionality for representing a single cron expression.

class crython.expression.**CronExpression**(*second*, *minute*, *hour*, *day*, *month*, *weekday*, *year*, *reboot=False*, *logger=None*)

Represents an entire cron expression.

An expression consists of seven, space delimited fields that represent the following values:

```
+------------- second (0 - 59)
| +------------- minute (0 - 59)
| | +------------- hour (0 - 23)
| | | +------------- day (1 - 31)
| | | | +------------- month (1 - 12)
| | | | | +------------- weekday (0 - 6) (Sunday to Saturday; 7 is also Sunday)
| | | | | | +------------- year (1970 - 2099)
| | | | | | |
| | | | | | |
* * * * * * *
```

classmethod new(*expression=None*, ***kwargs*)

Create a *CronExpression* instance from the given expression string or field values.

Parameters

- **expression** – (Optional) A string that can be converted to a cron expression.
- **kwargs** – (Optional) A dict that maps field names to values.

Returns A *CronExpression* that represents the given string or field values.

classmethod from_str(*expression*, *reboot_sentinel=<object object>*)

Create a *CronExpression* instance from the given cron expression string.

Parameters

- **expression** – A string that can be converted to a cron expression.
- **reboot_sentinel** – (Optional) Object that indicates the expression is a reboot; Default: REBOOT_SENTINEL

Returns A *CronExpression* that represents the given string.

classmethod from_kwargs(***kwargs*)

Create a *CronExpression* instance from the given dict of field name -> field value.

Parameters **kwargs** – A dict that maps field names to values.

Returns A `CronExpression` that represents the given dict.

classmethod from_reboot()

Create a `CronExpression` instance that indicates it's a "reboot" expression. A "reboot" expression means that it should be executed during startup and as soon as possible.

Returns A `CronExpression` that represents a "reboot".

property is_reboot

Return *True* if this expression represents a reboot; *False* otherwise.

matches(*dt*)

Check to see if the the given `datetime` instance "matches" this cron expression.

Parameters **dt** – A `datetime` instance to compare against.

Returns *True* if matches this expression; *False* otherwise.

1.2 crython/field

Contains functionality for representing an individual field within an expression.

class crython.field.**CronField**(*value*, *name*, *min*, *max*, *specials*)

Represents an individual field of a cron expression.

classmethod new(*value*, *name*, **args*, ***kwargs*)

Create a new `CronField` instance from the given value.

Parameters

- **value** – Value to create field from.
- **name** – Name of the column this field represents within an expression.
- **args** – Additional positional args
- **kwargs** – Additional keyword args

Returns A `CronField`

classmethod from_number(*value*, *name*, *min*, *max*, *specials*, **args*, ***kwargs*)

Create a new `CronField` instance from the given numeric value.

Parameters

- **value** – A `int` value.
- **name** – Name of the column this field represents within an expression.
- **min** – Lower bound for the value, inclusive
- **max** – Upper bound for the value, inclusive
- **specials** – Set of special characters valid for this field type
- **args** – Additional positional args
- **kwargs** – Additional keyword args

Returns A `CronField`

classmethod from_str(*value*, *name*, *min*, *max*, *specials*, **args*, ***kwargs*)

Create a new `CronField` instance from the given string value.

Parameters

- **value** – A `str` value.
- **name** – Name of the column this field represents within an expression.
- **min** – Lower bound for the value, inclusive
- **max** – Upper bound for the value, inclusive
- **specials** – Set of special characters valid for this field type
- **args** – Additional positional args
- **kwargs** – Additional keyword args

Returns A *CronField*

classmethod from_iterable(*value*, *name*, *min*, *max*, *specials*, **args*, ***kwargs*)
Create a new *CronField* instance from the given `Iterable` value.

Parameters

- **value** – A `Iterable` value.
- **name** – Name of the column this field represents within an expression.
- **min** – Lower bound for the value, inclusive
- **max** – Upper bound for the value, inclusive
- **specials** – Set of special characters valid for this field type
- **args** – Additional positional args
- **kwargs** – Additional keyword args

Returns A *CronField*

matches(*item*)
Check to see if the given time is 'within' the "time" denoted by this individual field.

..todo:: Recomputing this isn't very efficient. Consider converting the field or expression to a *datetime* or *timedelta* instance.

`crython.field.`**second**(*value*, ***, *name='second'*, ***kwargs*)
Partial for creating a `CronField` that represents the "second".

`crython.field.`**minute**(*value*, ***, *name='minute'*, ***kwargs*)
Partial for creating a `CronField` that represents the "minute".

`crython.field.`**hour**(*value*, ***, *name='hour'*, ***kwargs*)
Partial for creating a `CronField` that represents the "hour".

`crython.field.`**day**(*value*, ***, *name='day'*, ***kwargs*)
Partial for creating a `CronField` that represents the "day of month".

`crython.field.`**month**(*value*, ***, *name='month'*, ***kwargs*)
Partial for creating a `CronField` that represents the "day of month".

`crython.field.`**weekday**(*value*, ***, *name='weekday'*, ***kwargs*)
Partial for creating a `CronField` that represents the "day of week".

`crython.field.`**year**(*value*, ***, *name='year'*, ***kwargs*)
Partial for creating a `CronField` that represents the "year".

`crython.field.`**partials**
Mapping of field name to the partial that create one of that field "type".

1.3 crython/job

Contains functionality for defining functions that should be executed.

`crython.job.`**`job`**`(`**args*, ***kwargs*`)`

Decorate functions to execute them in the background at a scheduled time.

Parameters

- **`args`** – Positional args to pass to the decorated function.
- **`kwargs`** – Keyword args that contain job configuration and values to be passed to the decorated function.

1.4 crython/tab

Contains functionality for executing jobs (python functions) from cron expressions.

`class` `crython.tab.`**`CronTab`**`(`**args*, ***kwargs*`)`

Background thread responsible for executing background jobs.

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

`register``(`*name*, *job*`)`

Register the given name and function.

Parameters

- **`name`** – Name of the registered job. **Note:** This should be unique.
- **`job`** – Callable decorated with `job()` to execute.

Returns *None*.

`deregister``(`*name*`)`

De-register the job that was registered with the given name.

Parameters **`name`** – Name of the job to remove.

Returns *None*.

`stop``()`

Stop this background thread from executing any more jobs.

Returns *None*.

`run``()`

Background function that processes all registered jobs and invokes them based on their context and expression.

`crython.tab.`**`default_tab`**
The default, global tab instance that is created on import. This is the instance that will be used unless the *`job()`* caller overrides it.

`crython.tab.`**`start`**`()`
Start the default, global *`CronTab`* instance.

`crython.tab.`**`stop`**`()`
Stop the default, global *`CronTab`* instance.

This informs the background thread to stop processing new jobs but does not wait for it to finish completing its current ones. If the caller wishes to wait for the thread to stop completely, it should call `join()`.

PYTHON MODULE INDEX

INDEX