timeparser Documentation

Release 0.7.2

thomst

July 25, 2013

CONTENTS

Contents:

CHAPTER

ONE

TIMEPARSER

Parse strings to objects of the datetime-module.

This module intends to make string-parsing to objects of the datetime-module as easy as possible while allowing a fine configuration about which kind of formats are supported.

1.1 Latest Version

The latest version of this project can be found at : http://github.com/thomst/timeparser.

1.2 Installation

- Option 1 : Install via pip

  ```
  pip install timeparser
  ```

- Option 2 : If you have downloaded the source

  ```
  python setup.py install
  ```

1.3 Dokumentaion:

Please visit the documentation on readthedocs.org: https://timeparser.readthedocs.org/en/latest/index.html

1.4 Usage

How to use?

```
>>> import timeparser
>>>
>>> timeparser.parsedate('24.4.13')
datetime.date(2013, 4, 24)
>>>
>>> timeparser.parsedate('24 Apr 2013')
datetime.date(2013, 4, 24)
>>>
```

```
>>> timeparser.parsetime('234405')
datetime.time(23, 44, 5)
>>>
>>> timeparser.TimeFormats.config(allow_no_sep=False)
>>> timeparser.parsetime('234405')
ValueError: couldn't parse 234405 as time
>>>
>>> timeparser.parsedatetime('24-04-13_23:44:05')
datetime.datetime(2013, 4, 24, 23, 44, 5)
```

1.5 Changes in v0.7

The formats-classes accept an keyword try_hard, which means they try to build formats for what ever kind of string was passed -regardless of any configuration

1.6 Reporting Bugs

Please report bugs at github issue tracker: https://github.com/thomst/timeparser/issues

1.7 Author

thomst <thomaslfuss@gmx.de> Thomas Leichtfuß

- http://github.com/thomst

CHAPTER

TWO

DOKUMENTATION

Parse strings to objects of `datetime`.

This module intends to make string-parsing to `datetime`-objects as easy as possible while allowing a fine configuration about which kind of formats are supported:

Parsing any kind of string is as easy as:

```
>>> date = parsedate('3 Jan 2013')
datetime.date(2013, 1, 3)
```

Now suppose you don't want to allow parsing strings with literal month-names:

```
>>> DateFormats.config(allow_month_name=False)
>>> date = parsedate('3 Jan 2013')
ValueError: couldn't parse '3 Jan 2013' as date
```

Most of the time you will use format-classes only to alter their configuration. The parser-functions (except `parsetimedelta()`) use the format-classes to recieve a list of format-strings and try to parse the string with them using `datetime.datetime.strptime()`.

`parsetimedelta()` breaks with that concept. It don't need format-strings at all and has his own `logic`.

2.1 A closer look at format-classes

Format-classes are actual `list`-types that provides two main-features:

- They produce themselves as lists of format-strings accordingly to a set of parameters,
- and they are configurable in regard to these parameters.

To create a list with an altered configuration you can either pass keyword- arguments to the constructor:

```
>>> formats = TimeFormats(seps=['-', ':', ';'], allow_microsec=True)
```

or change the default-configuration on class-level:

```
>>> TimeFormats.config(seps=['-', ':', ';'], allow_microsec=True)
>>> formats = TimeFormats()
```

Both will result in the same list of formats, but the former way doesn't touch the default-configuration.

If you just call the constructor the format-class will produce a list of all formats for the actual configuration:

```
>>> formats = DateFormats()
>>> len(formats)
77
```

But if you look for formats for a specific string you can pass the string to the constructor:

```
>>> DateFormats('3 Jan 2013')
['%d %b %Y']
```

That is what the parser-functions do to minimize the amount of formats they have to try to parse the string with.

Producing formats for a specific string also respects the current setting:

```
>>> set(DateFormats('3 Jan 2013')) < set(DateFormats())
True
>>> DateFormats.config(allow_month_name=False)
>>> DateFormats('3 jan 2013')
ValueError: no proper format for '3 jan 2013'
```

2.2 Parser-functions

timeparser.**parsetime**(*string*, *formats*=[ ])

Parse a string to a `datetime.time` -object.

Parameters

- **string** (*str*) – String to be parsed.
- **formats** (*list*) – Optional list of formats-string.

Return type `datetime.time`

Raises ValueError, if string couldn't been parsed

The string is tried to be parsed with every format of *formats*. If *formats* not given `TimeFormats`(string) is used.

---

timeparser.**parsedate**(*string*, *formats*=[ ], *today*=*None*)

Parse a string to a `datetime.date`-object.

Parameters

- **string** (*str*) – String to be parsed.
- **formats** (*list*) – Optional list of formats-string.
- **today** (*datetime.date*) – optional date

Return type `datetime.date`

Raises ValueError, if string couldn't been parsed

string is tried to be parsed with every format of *formats*. If *formats* not given `DateFormats`(string) is used.

If *string* is parsed with an incomplete format (missing year or year and month), the date will be completed by *today* or `timeparser.TODAY`.

---

timeparser.**parsedatetime**(*string*, *formats*=[ ], *today*=*None*)

Parse a string to a datetime.datetime-object.

Parameters

- **string** (*str*) – String to be parsed.
- **formats** (*list*) – Optional list of formats-string.
- **today** (*datetime.datetime*) – Optional date

Return type datetime.datetime

Raises ValueError, if string couldn't been parsed

string is tried to be parsed with every format of *formats*. If *formats* not given DatetimeFormats(string) is used.

If *string* is parsed with an incomplete format (missing year or year and month), the date will be completed by *today* or timeparser.TODAY.

---

timeparser.**parsetimedelta**(*string*, *key*='*weeks*')

Parse a string to a datetime.timedelta-object.

Parameters

- **string** (*str*) – String to be parsed.
- **key** (*str*) – String that contains or matches a timedelta-keyword (defaults to 'weeks').

Return type datetime.timedelta

Raises ValueError, if string couldn't been parsed

parsetimedelta looks for digits in *string*, that could be seperated. These digits will be the arguments for datetime.timedelta. Thereby *key* is used to determine the *unit* of the first argument, which could be one of the keywords for datetime.timedelta ('weeks', 'days', 'hours', 'minutes', 'seconds'). The following arguments get each the next lesser *unit*:

```
>>> parsetimedelta('1, 2, 3', 'h') == datetime.timedelta(hours=1, minutes=2, seconds=3)
True
```

Another way is to just place keyword-matching literals within the string:

```
>>> parsetimedelta('1h 2m 3s') == datetime.timedelta(hours=1, minutes=2, seconds=3)
True
```

2.3 Format-classes

class timeparser.**TimeFormats**(*string*=*None*, *seps*=*None*, *allow_no_sep*=*None*, *figures*=*None*, *try_hard*=*None*, *use_formats*=*None*, *use_sformats*=*None*)

A list of time-string-formats that generates himself.

Parameters

- **string** (*str*) – Pre-select formats for string.
- **seps** (*list*) – Allowed separators for formats.
- **allow_no_sep** (*bool*) – Allows formats without any separator.
- **figures** (*list*) – List of four booleans (s. FIGURES).

- **try_hard** – Regardless of any configuration try hard to build formats for the given string.

Raises ValueError if no format could be produced for *string*.

SEPS = [':', ' ']
A list of separators, formats are produced with.

ALLOW_NO_SEP = True
Allows formats without any separator ('%H%M%S').

FIGURES = [True, True, True, False]
List of four booleans that predicts how many digits formats are allowed to have:

•figures[0]: Allows the one-digit format '%H'.

•figures[1]: Allows two-digit-formats like '%H:%M'.

•figures[2]: Allows three-digit-formats like '%H:%M:%S'.

•figures[3]: Allows four-digit-formats like '%H:%M:%S.%f'.

---

class timeparser.**DateFormats**(**args*, ***kwargs*)
A list of date-string-formats that generates himself.

Parameters

- **string** (*str*) – Pre-select formats for string.
- **seps** (*list*) – Allowed separators for formats.
- **allow_no_sep** (*bool*) – Allows formats without any separator.
- **figures** (*list*) – List of three booleans (s. FIGURES).
- **allow_month_name** (*bool*) – Allows formats with month-names (%b or %B)
- **try_hard** – Regardless of any configuration try hard to build formats for the given string.

Raises ValueError if no format could be produced for *string*.

SEPS = ['.', '-', '/', ' ', '. ']
A list of separators, formats are produced with.

ALLOW_NO_SEP = True
Allows formats without any separator ('%d%m%y').

FIGURES = [True, True, True]
List of three booleans that predicts how many digits formats are allowed to have:

•figures[0]: Allows the one-digit format '%d'.

•figures[1]: Allows two-digit-formats like '%d/%m'.

•figures[2]: Allows three-digit-formats like '%d/%m/%y'.

classmethod config(**args*, ***kwargs*)
Modify class-configuration.

Parameters

- **seps** (*list*) – Allowed separators for formats.
- **allow_no_sep** (*bool*) – Allows formats without any separator.
- **figures** (*list*) – List of three booleans (s. FIGURES).
- **allow_month_name** (*bool*) – Allows formats with month-names (%b or %B)

- **try_hard** – Regardless of any configuration try hard to build formats for the given string.

---

class timeparser.**DatetimeFormats**(**args*, ***kwargs*)
A list of datetime-string-formats that generates himself.

Parameters

- **string** (*str*) – Pre-select formats for string.
- **seps** (*list*) – Allowed separators for formats.
- **allow_no_sep** (*bool*) – Allows formats without any separator.
- **date_config** (*dict*) – kwargs `DateFormats` are initialized with
- **time_config** (*dict*) – kwargs `TimeFormats` are initialized with
- **try_hard** – Regardless of any configuration try hard to build formats for the given string.

Raises ValueError if no format could be produced for *string*.

SEPS = [' ', '.', '_', ':']
A list of separators, formats are produced with.

ALLOW_NO_SEP = True
Allows formats without any separator ('%H%M%S').

classmethod config(**args*, ***kwargs*)
Modify class-configuration.

Parameters

- **seps** (*list*) – Allowed separators for formats.
- **allow_no_sep** (*bool*) – Allows formats without any separator.
- **date_config** (*dict*) – kwargs `DateFormats` are initialized with
- **time_config** (*dict*) – kwargs `TimeFormats` are initialized with
- **try_hard** – Regardless of any configuration try hard to build formats for the given string.

2.4 Endianness and Date-completition

timeparser.**ENDIAN = ('day', 'month', 'year')**

In generell dates could have one of three orders:

- little-endian: *day, month, year*
- big-endian: *year, month, day*
- middle-endian: *month, day, year*

ENDIAN is an instance of `Endian` and defines the order that should be applied:

```
>>> ENDIAN
('day', 'month', 'year')
>>> parsedate('26/4/13')
datetime.date(2013, 4, 26)
```

On creation a local-default-order is guessed, but could be changed through `Endian.set()`:

```
>>> ENDIAN.set('big')
>>> ENDIAN
('year', 'month', 'day')
>>> parsedate('26/4/13')
datetime.date(2026, 4, 13)
```

Warning: Guessing the local default is in a provisional state and a middle-endian- order is not regarded at all.

class `timeparser.`**`Endian`**

Endian emulates a tuple, which represents the order of a date.

Dates can be ordered in three different ways:

- little-endian: `('day', 'month', 'year')`
- big-endian: `('year', 'month', 'day')`
- middle-endian: `('month', 'day', 'year')`

On creation a local default-order is guessed (either little- or big-endian). To change it use `set()`.

`set`(*key=None*)

Set ENDIAN to little-, big- or middle-endian.

Parameters **key** (*str or None*) – A string matching 'little', 'big' or 'middle'.

If key is None the local-default-order is guessed.

`timeparser.`**`TODAY = datetime.date(2013, 7, 25)`**

TODAY is an instance of `Today` and is used to complement dates that were parsed with an incomplete format-string:

```
>>> TODAY
TODAY(2013, 5, 9)
>>> parsedate('20 Apr')
datetime.date(2013, 4, 20)
```

or even:

```
>>> TODAY
TODAY(2013, 5, 9)
>>> parsedate('20')
datetime.date(2013, 5, 20)
```

TODAY defaults to `datetime.date.today()`, but can be changed through `Today.set()`:

```
>>> TODAY.set(2000, 1, 1)
>>> parsedate('20')
datetime.date(2000, 1, 20)
```

class `timeparser.`**`Today`**

Today emulates a `datetime.date`-object that could be changed through `set()`.

On creation Today will be set to `datetime.date.today()`.

Because `datetime.date`-objects are not mutable (but Today-instance has to be), Today imitates a `datetime.date` just saving one as `Today.dateobj` and let `Today.year`, `Today.month` and `Today.day` returning its values.

set (**args*, ***kwargs*)
Change TODAY.

Parameters

- **year** (*int*) – year
- **month** (*int*) – month
- **day** (*int*) – day

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *search*

PYTHON MODULE INDEX

t