Events webscraping

Release 0.1 - 2019-11-07

Vincent Férotin

Nov 07, 2019

CONTENTS:

CHAPTER

ONE

INSTALLATION

1.1 Requirements

This project works on and is tested on Python version "3.5". It requires following tools:

Tool	Version
Scrapy	1.5
PyYAML	3.12
Unidecode	1.0.22

Note: A note about version numbers

While above version numbers seems precises, they are just current versions of each components upon which the project is built on. These versions are those tested on development, so "known to work". Perhaps previous versions would work fine too…

1.2 Getting the sources

Project is currently developped at the *ENS CRI* forge (a *GitLab* instance), at: https://forge.cri.ens.fr/risc/events-webscrapping using git as version control system.

You could clone the repository with:

```
$ cd /home/work/RISC/events
$ git clone https://<yourname>@forge.cri.ens.fr/risc/events-webscrapping.git web-
↪scraping
```

1.3 Installation of a working environment on *Debian* testing

1. First, ensure there is some version of Python 3 already installed on your system:

   ```
   $ python3 --version
   Python 3.6.5rc1
   ```

2. Then, you need Python 3.5:

   ```
   $ apt install python3.5 python3.5-dev
   ```

3. Then, you also need a recent version of pip for Python 3:

```
$ apt install python3-pip
# Test installation:
$ python3 -m pip --version
# Update pip and setuptools them-selves:
$ python3 -m pip install --user --update setuptools
$ python3 -m pip install --user --update pip
```

4. Then, install pipenv:

```
$ python3 -m pip install --user pipenv
```

   Now *pipenv* should be available in your `~/.local/bin/` directory:

```
$ ~/.local/bin/pipenv --version
```

5. For convenience, you could add this directory to your *$PATH* (e.g. in your `~/.bashrc` or equivalent):

```
$ export PATH="$HOME/.local/bin:$PATH"
```

6. Last, initialize a new *virtualenv* managed by *pipenv*, in source folder, with all project dependencies:

```
$ cd /home/work/RISC/events/web-scraping/
$ pipenv install
```

   If your goal is to hack on the project, add `--dev` option:

```
$ pipenv install --dev
```

CHAPTER

TWO

OVERVIEW

Present project, named *events*, is both a *framework* on top of *Scrapy* and the *crawler* of websites to scrape. In order to develop it, you must first understand how *Scrapy* works.

2.1 Scrapy overview

Scrapy lets you define *spiders* for specifying how to parse and extract information of a given site, in so called `spiders` module. *Spiders* are run by a *crawler* created in project's `main()` function.

A *spider* is usually specific to one website, or a set of very similar websites (in their structures). It is defined in a class derivating from `scrapy.Spider`, specifies a set of starting URLs, and a `parse()` callback method to set the rules to apply to all of these URLs. This `parse()` method takes an HTTP *Response* object as parameter, the one generated by *GET*ting one of these starting URLS, and let use it to generate (yielding) *items*.

Those *items* are structured pieces of information, usually extracted from body of the *Response*, thanks *CSS* or *XPath* selectors, and stored as objects of class derivating from `scrapy.Item` (usually standing on `items` folder). They are then passed to the item *pipeline*, which post-processes them, in order for e.g. validating data, and storing item to database.

See also:

https://docs.scrapy.org/en/latest/topics/architecture.html#data-flow for an overview diagram of *Scrapy* architecture and messages process.

Although this big picture seems understandable, there is one more concept to grab: before an *item* is fullfilled with data extracted from *Response*, this data walk through an *item loader*. A *loader* defines, for each *field* of the *item*, an *input processor* and an *output processor*, where the *input processor* collects and pre-processes data which could be used to create the *field*'s value, and the *output processor* processes this whole collection in order to "summarize" it into a single value, the one finally affected to the field.

2.2 Additions of present project

One core addition of this project to *Scrapy* is the idea (and related tools) that, from *XPath* extraction to final storage as an *Item*, processing data could fail at any step. Instead of relying of the logging mechanism to detect such failures, we argue that it is easier to report failure in "lieu et place" in the given *item*. To achieve this goal, we let failure occure, attach some description about it, and let this package "bubble" through the rest of the process – given that subsequent processing let this package as is. Such a package is offered by our *ParsableValue* class.

Also, we introduce the notion of *preprocessor*: it is a pipeline, like the *input processor* of an *item loader*, processing some data *before* it is even passed to an *item loader*. It starts with e.g. *XPath* extraction, and ends where the *item loading* starts. Of course, its output is also a *ParsableValue*.

CHAPTER

THREE

NARRATIVE DOCUMENTATION

3.1 Webscrapping tutorial with dedicated components

This tutorial is not an introduction to webscrapping with Scrapy, but is dedicated to particluarities of using present facilities added on top of *Scrapy*.

3.1.1 Declaring *Item* and its *Loader* classes

The goal of webscrapping with *Scrapy* is to fulfill some *Item*s from downloaded pages. Declaring them is identical to the *Scrapy* way, but their related loaders are slightly differents.

Item Loader must inherit not from `scrapy.loader.ItemLoader` but from *`items.base.ItemLoader`*. Besides this custom inheriting, declaration follows *Scrapy* way. Note however that processors must be picked from `items.processors`. Also, it is recommanded to explicitely define the `default_item_class` class attribute linking the loader class to its item class, e.g.:

Listing 1: items/person.py

```
from scrapy.loader.processors import Identity, MapCompose
from scrapy import Item, Field

from .base import ItemLoader
from .processors import TakeFirstParsed, init, remove_tags, str_strip

class PersonItem(Item):
    name = Field()

class PersonLoader(ItemLoader):
    default_item_class = PersonItem

    default_input_processor = Identity()
    default_output_processor = TakeFirstParsed()

    name_in = MapCompose(init, remove_tags, str_strip)
```

See also:

`items.base.ItemLoader A.P.I.`

3.1.2 Declaring *Item Loader* processors

Since we use *parsability and bubbling*, we *must* use project's *processors*. The first one is of course *`init`*, to wrap initial value into a *`ParsableValue`*:

```
from scrapy.loader.processors import MapCompose

from .base import ItemLoader
from .processors import init, other_proc

class PersonLoader(ItemLoader):
    name_in = MapCompose(init, other_proc)
```

`scrapy.loader.processors.MapCompose` is used here to apply a serie of input processors to all values the *loader* will receive for the *field*.

One particular processor is `TakeFirstParsed`, which is usually used as default output processor in `default_output_processor`:

```
from scrapy.loader.processors import Identity

from .base import ItemLoader
from .processors import TakeFirstParsed

class PersonItem(Item):
    name = Field()

class PersonLoader(ItemLoader):
    default_output_processor = TakeFirstParsed()
```

whereas default input processor (`default_output_processor`) is generally `scrapy.loader.processors.Identity`:

```
from scrapy.loader.processors import Identity

from .base import ItemLoader

class PersonLoader(ItemLoader):
    default_input_processor = Identity()
```

See also:

- *introduction on processors*,
- *items.processors A.P.I.*.

3.1.3 Starting writing a spider

When our *Item* classes and their related loaders classes are declared, we could start writing our first spider.

Listing 2: spiders/org_domain_www.py

```
from scrapy import Spider

class OrgDomainWww(Spider):
    name = "www.domain.org/index.html"
    start_urls = [
        'http://www.domain.org/index.html',
    ]

    def parse(self, response):
        # Do something interesting...
```

3.1.4 Preprocessing data before passing it to item loader

Before passing data to the *processors* pipeline of *item loader*s, it is common to *preprocess* them, between *XPath* extraction and passing them to first *processor*, through a defined *pipeline*. In this *pipeline*, *parsability and bubbling* are equally applied.

Our first *preprocessor* will usually be `XPathExtractor`. If it is the only preprocessor to apply, we could solely call it:

Listing 3: spiders/org_domain_www.py

```
from ..items.preprocessors import XPathExtractor
from ..items.parsability import ParsableValue

class OrgDomainWww(Spider):
    #...

    def parse(self, response):
        # Parse title
        preprocessor = XPathExtractor('//article[@id="post-268"]'
                                      '/div[@class="entry-content"]'
                                      '/p'
                                      '/text()',
                                      'extract_first')
        title = preprocessor(response)
        assert isinstance(title, ParsableValue)
        # ... continue scrapping
```

Instead, if additional preprocessors need to be applied, such as `StrSplitter`, `IndexSelector`, and `TagsRemover`, we should better construct a *pipeline*:

Listing 4: spiders/org_domain_www.py

```
from ..items.preprocessors import PreprocessorPipeline, XPathExtractor,␣
↪StrSplitter, IndexSelector, TagsRemover
from ..items.parsability import ParsableValue

class OrgDomainWww(Spider):
    #...

    def parse(self, response):
        # ... title is parsed
        # parse a person
        preproc_pipeline = PreprocessorPipeline(
            XPathExtractor((
                    '//article[@id="post-268"]'
                    '/div[@class="entry-content"]'
                    '/p[[text()="{}"]'
                    '/parent::*'
                    ).format(title.value),
                'extract_first'),
            StrSplitter('<br>'),
            IndexSelector(1),
            TagsRemover())
        person = preproc_pipeline(response)
        assert isinstance(person, ParsableValue)
```

Of course, it is also feasible as a shortcut to directly call preprocessor or preprocessors pipeline with *response* as attribute, just after initializing it with our *XPath*:

Listing 5: spiders/org_domain_www.py

```
title = XPathExtractor(
    '//article[@id="post-268"]'
    '/div[@class="entry-content"]'
    '/p'
    '/text()',
    'extract_first')\
    (response)
assert isinstance(title, ParsableValue)
```

See also:

- *introduction on preprocessors*,
- *items.preprocessors A.P.I.*.

3.1.5 *Items loading*

Since we successfully preprocessed our data, we could now fullfill *items*. Our data is still an *HTML* snippet, so we could use our custom *ItemLoader.add_regexp* to extract e.g. name from a link:

Listing 6: spiders/org_domain_www.py

```
import re

from ..items.person import PersonItem, PersonLoader

PERSON_RE = re.compile(
    r'^'
    r'\s*(\\n)?\s*'
    r'<a href=".*">\s*'
    r'(?P<name>.*)'
    r'</a>\s*'
    r'.*'
    r'$'
    )
"""Parse a person linked to an URL (:class:`re.RegExp`)."""

class OrgDomainWww(Spider):
    #...

    def parse(self, response):
        # ...
        # assumed that persons are already preprocessed and well parsed
        for person in persons.value:
            person_loader = PersonLoader(item=PersonItem())
            person_loader.add_regexp(PERSON_RE, person, {
                'name': 'name',
                })
            # Create and store Person
            person = person_loader.load_item()
            yield person
```

See also:

items.base.ItemLoader A.P.I.

3.2 *ParsableValue* class and bubbling

One of the additions this project builds on top of *Scrapy* is the notion of *parsability*. By default a "value", during the process from its *XPath* extraction to its store in an *Item*, is transformed by several tools. Each transformation *could* eventually fail, and the only way to detect that is to read the detailed log and hope that you had reported failure here.

It should be better to have failure report in the final *Item* structure, with additional infos about it. So, instead of passing "naïve" values between e.g. processors, we could imagine wrapping these values with some metadata about their state regarding parsing. This is the purpose of *items.parsability.ParsableValue*.

Some of this class' instances wraps any value in an *immutable* structure, letting access to it through its *.value* attribute. But it also stores a state flagging if the parsing process went well or not, through its *.is_parsed()* method: getting a `False` implies that process failed at some point.

Aside wrapping value with metadata about *parsability*, we also *do not process* already unparsed values, letting a processor pass it as-is to its neighbour, so that unparsed value "bubble" next to its *Item Field*, so final user could see unparsability infos in "lieu et place" of well parsed value in final *Item* structure.

A *items.parsability.ParsableValue* is built through its initializer with two options:

- if the *value* is well parsed, a new *ParsableValue* is simply built as is: `ParsableValue(value)`;
- else, if it was not well parsed, a new *ParsableValue* is initialized with `None` as its value, and additional infos like a "reason" and previously unparsed value: `ParsableValue(None, unparsed_reason="XXX", unparsable_value=value)`.

If a value *is not parsed*, additional informations could be get by properties *.unparsed_reason* and *.unparsable_value*.

ParsableValue and *bubbling* are used in *Item Loader* *processors*, of course but also in *preprocessors*.

See also:

items.parsability.ParsableValue A.P.I.

3.3 *Item Loader* processors

Before a *Field* is fullfilled with a well parsed value, this value will pass to a serie of processors defined by the loader of the *Item* class. Because of using *parsability and bubbling*, we have to define our custom *processors*.

The first is obviously *items.processors.init*, which wrap starting value in a *ParsableValue*.

Other available processors are:

- *int*,
- *remove_tags*,
- *str2date*,
- *str2time_duration*,
- *str_capitalize*,
- *str_strip*,
- *str_unquote*,
- *str_unword*.

A particular processor is *TakeFirstParsed*, which is usually used as `default_output_processor` of the loader.

See also:

- *tutorial's section on processors*,
- *items.processors A.P.I.*.

3.4 "*Preprocessors*" and their pipelines

Before some value could be passed to the *Item Loader processors*, we would like to get a similar pipeline between data extraction from *XPath* selector and firs input *processor* in *item loader*. Moreover, it could be easier to debug spiders if we also could use *ParsableValue* at this first steps of data processing. This is the intent of *preprocessors*.

A *processors pipeline* is a serie of *processors* to manipulate data from its first extraction step, usually from a *XPath*. It is defined in `items.preprocessors` in `PreprocessorPipeline`. A *pipeline* is initialized with a serie of *preprocessors*, either by its `__init__` or `add` methods, and then `called` with initial value to be passed to the *pipeline*:

```
from .items.preprocessors import PreprocessorPipeline, Preproc1, Preproc2

parsable_value = PreprocessorPipeline(
    Preproc1(**params1),
    Preproc2(**params2),
    )(initial_value)
```

First preprocessors are usually `XPathExtractor` and `MultiXPathExtractor`. Then any combination of other preprocessors could be used to build the pipeline, with:

- `IndexSelector`,
- `RegExpExtractor`,
- `StrSplitter`,
- `TagsRemover`.

Once preprocessed, the initial value, now a `ParsableValue`, if `already parsed`, could be passed "as is" to its related *Item Loader processors*.

3.5 Running crawlers

Once spiders are declared (see *Webscrapping tutorial with dedicated components*), ensure that they are also nammed in `events/__init__.py` module in `events.SPIDERS`. Then crawling could be called through command line, in software project's root directory:

```
$ python -m events [options]
```

There is 3 files as software's output:

- a SQLite database where items are stored;
- a YAML dump file for unparsable items' values;
- a YAML dump file for well parsed items' values.

Each could be set either by *configuration file* or by *arguments passed through command-line*. Value order for option are taken from:

1. `default values`;
2. *configuration file*;
3. *arguments passed through command-line*;

so that each value passed as argument is taken over each value defined in configuration file, which in turn has precedence over default ones.

3.5.1 Configuration file

Configuration file is Scrapy's one: `scrapy.cfg` at the root of the project. Three new options are defined in `[settings]` section:

db_path Path of file used by *SQLite* as database, where scrapped items will be stored.

> **Warning:** Already existing file will be deleted!
>
> If this file exists before running crawlers, it will be destroyed (and then recreated from scratch): all previously stored data will be lost.

unparsed_dump Path of *YAML* file used as dump for unparsed items.

> **Warning:** Already existing file will be deleted!
>
> If this file exists before running crawlers, it will be destroyed (and then recreated from scratch): all previously stored data will be lost.

parsed_dump Path of *YAML* file used as dump for well parsed items.

> **Warning:** Already existing file will be deleted!
>
> If this file exists before running crawlers, it will be destroyed (and then recreated from scratch): all previously stored data will be lost.

Example of `scrapy.cfg`

```
[settings]
default = events.settings
db_path = events.db.sqlite3
unparsed_dump = parsing_errors.yaml
parsed_dump = parsed_items.yaml
```

3.5.2 Arguments on command-line

Command-line options are the same as these defined in *configuration file*:

--db_path Path of file used by *SQLite* as database, where scrapped items will be stored.

> **Warning:** Already existing file will be deleted!
>
> If this file exists before running crawlers, it will be destroyed (and then recreated from scratch): all previously stored data will be lost.

--unparsed_dump Path of *YAML* file used as dump for unparsed items.

> **Warning:** Already existing file will be deleted!
>
> If this file exists before running crawlers, it will be destroyed (and then recreated from scratch): all previously stored data will be lost.

--parsed_dump Path of *YAML* file used as dump for well parsed items.

> **Warning:** Already existing file will be deleted!
>
> If this file exists before running crawlers, it will be destroyed (and then recreated from scratch): all previously stored data will be lost.

Example of command

```
$ python -m events \
    --db_path=~/scrapped_items.sqlite \
    --unparsed_dump=/dev/null \
    --parsed_dump=~/scrapped_items.yaml
```

CHAPTER

FOUR

APPLICATION PROGRAMMING INTERFACE (*A.P.I*)

4.1 events

Main module of events scrapping.

events.**DEFAULT_DATABASE_PATH = 'events.db.sqlite3'**
Name of default database where storing scrapped items (*str*).

events.**DEFAULT_PARSED_DUMP = 'parsed_items.yaml'**
Name of default dump file path for well parsed items (*str*).

events.**DEFAULT_UNPARSED_DUMP = 'parsing_errors.yaml'**
Name of default dump file path for errors of parsing (*str*).

events.**SCRAPY_CONFIG = 'scrapy.cfg'**
Name of default Scrapy configuration for deployment (*str*).

events.**SPIDERS = ('calcul.hypotheses.org/programme-2016-2017', 'cerca.labo.univ-poitiers**
List of spiders (names) to crawl (*t-uple* of *str*).

events.**create_database**(*db_path='events.db.sqlite3'*)
Create a fresh new SQLite3 database, in lieu et place of any existing one.

Parameters **db_path** (*str*) – Path of new SQLite database.

events.**get_config**(*cli_args*, *cli_arg_name*, *ini_config*, *ini_section*, *ini_option*, *default_value*)
Get configuration value, from both CLI parsed args, INI config. and default one.

Priority is: default value < INI config < CLI args.

Parameters

- **cl_args** (argparse.Namespace) – CLI parsed arguments.
- **cli_arg_name** (*str*) – Option name in CLI parsed arguments.
- **ini_config** (configparser.ConfigParser) – INI parsed configuration.
- **ini_section** (*str*) – INI section of the option.
- **ini_option** (*str*) – INI option name.
- **default_value** – Default value.

Returns Configuration value from either default one, INI config. or parsed CLI arguments.

Return type *object*

events.**main**()
Main function, software entry point.

4.2 items

4.2.1 Utilities

items.base

Base Item Loader.

class events.items.base.**ItemLoader**(*item=None*, *selector=None*, *response=None*, *parent=None*, ***context*)

Custom item loader.

Its purpose is to manage values as *parsability.ParsableValue*. It also introduces new methods:

- *add_parsed_value_field()*,
- *add_regexp()*.

Its usage is the same as scrapy.loader.ItemLoader; your own items loaders should inherit from this class instead of Scrapy's one.

Example

Listing 1: items/person.py

```
from scrapy.loader.processors import Identity, MapCompose
from scrapy import Item, Field

from .base import ItemLoader
from .processors import TakeFirstParsed, init, remove_tags, str_strip

class PersonItem(Item):
    name = Field()

class PersonLoader(ItemLoader):
    default_input_processor = Identity()
    default_output_processor = TakeFirstParsed()

    name_in = MapCompose(init, remove_tags, str_strip)
```

Listing 2: spiders/myspider.py

```
from ..items import PersonItem, PersonLoader

class MySpider(Spider):

    def _parse_persons(self, response, full_title):
        person = response.xpath('//article[@id="post-268"]/p')\
                         .extract_first()
        if not person.is_parsed():
            return person
        # else:
        person_loader = PersonLoader(item=PersonItem())
        person_loader.add_regexp(PERSON_RE, person, {
            'name': 'name',
            })
        person = person_loader.load_item()
        return person
```

add_parsed_value(*field_name*, *parsed_value*, **processors*, ***kw*)

Add a parsed value.

Parameters **parsed_value** (*parsability.ParsableValue*) – Already parsed value, whose `value` attr. contains some accessible fields by `getattr()`.

add_parsed_value_field(*item_field*, *parsed_value*, *pv_field*)
Add a value from a field of an already parsed value.

Parameters

- **item_field** (*str*) – Item field name to which add this value.
- **parsed_value** (*parsability.ParsableValue*) – Already parsed value, whose `value` attr. contains some accessible fields by `getattr()`.
- **pv_field** (*str*) – Parsed value field name.

add_regexp(*regexp*, *value*, *fields*)
Add value from a regular expression.

Parameters

- **regexp** (`re.RegExp`) – Regular expression for extracting fields from `value`.
- **value** (*str*) – Value from which extract fields by `regexp`.
- **fields** (*dict* of *str*: *str*) – Item fields names mapped to their related `regexp` groups.

Returns **`dict` of `str`** – Extracted fields from `value`.

Return type (*str* or `None`)

add_value(*field_name*, *value*, **processors*, *unparsed_reason=None*, *unparsable_value=None*, ***kw*)
Custom *add_value()*, allowing to pass a reason if `value` is `None`.

Parameters

- **unparsed_reason** (*str*) – Reason for unparsability if `value` is None.
- **unparsable_value** (*object*) – Unparsable value if any.

items.parsability

Item parsability.

class events.items.parsability.**ParsableValue**(*value*, *parsed=None*, *unparsable_value=None*, *unparsed_reason=''*)
Custom value object, allowing storing metadata in case of unparsability.

Note: Immutable structure data!

A *ParsableValue* is by nature immutable. Once *initialized*, its attributes could not be changed, only accessed.

Its purpose is to allow "bubbling" unparsed valued to the final Item structure, so that scrapping/parsing could be modified easily thanks to readable message on process failure, instead of searching through (debug) logging messages.

To know if a value is well parsed, ask *is_parsed()*. In case it is `False`, you could get more infos by *unparsed_reason* and *unparsable_value*.

value
Value in parsing process – could be anything. Its value could be `None`, meaning that parsing process failed at one of previous stages – see `unparsable_infos()` in that case.

unparsed_reason
"Reason" of unparsability of *unparsable_value*.

Type str

unparsable_value
Value which failed to be processed – could be anything.

Example

```
from .processors import str_unquote, str_unword, str2date

date = ParsableValue("'24 Janvier 2017'")
date = str2date(str_unquote(str_unword(date)))
if date.is_parsed():
    date = date.value
else:
    print(date.unparsed_reason)
```

__copy__()
Shallow copy of instance.

__deepcopy__(*memo*)
Deep copy of instance.

__init__(*value*, *parsed=None*, *unparsable_value=None*, *unparsed_reason=''*)
Initialization.

Parameters

- **value** – Value to parse (see *value*)
- **parsed** (*bool*) – Flag telling if *value* is already well parsed.
- **unparsable_value** – Blob which failed to be parsed.
- **unparsed_reason** (*str*) – Description of unparsability, if any.

__repr__()
Return repr(self).

__weakref__
list of weak references to the object (if defined)

is_parsed()
Tell if the field was fully parsed.

Returns `True` if the field was successfuly parsed, `False` otherwise.

Return type *bool*

items.preprocessors

Preprocessors to parse value before loading items.

This module define a *PreprocessorPipeline*, and several preprocessors like *XPathExtractor*, *TagsRemover* and so on to use in a *pipeline*.

Available preprocessors are:

- *AttrsFiller*,
- *IndexSelector*,
- *MultiXPathExtractor*,
- *RegExpExtractor*,
- *StrSplitter*,

- *TagsRemover*,
- *XPathExtractor*.

The module also defines a *extract_raw_date()* utility.

class events.items.preprocessors.**AttrsFiller**(*attrs*)
Attribute filler.

attrs
Dict where all keys will be new attributes of parsed value, and whose values are callable used to construct new values.

Type *dict* of *str*: callable

Example

For single use (i.e. outside a pipeline):

```
filler = Attrsdatetime.htmlFiller({
    'year': lambda x: x.year,  # identity
    'month': lambda x: month_number(x.month),
    'note': lambda x: 'Fixed' if x.year % 2 else 'Raw',  # Add a random note
    })
value = filler(value)
# value has now a 'year' attr., valued to previous attr. 'year';
# a 'month' attr. valued to an `int` computed from previous repr. of month,
# and an additional note.
```

__call__(*value*)
Run filler.

Parameters **value** (*object*) – Object on which operate filling of attributes.

Returns Parsed value, whose value attribute contains all attributes defined in *attrs*, and whose values are computed from value's ones passed to *attrs* callables.

Return type *parsability.ParsableValue*

__init__(*attrs*)
Initialization.

Parameters **attrs** (*dict* of *str*: callable) – Dict where all keys will be new attributes of parsed value, and whose values are callable used to construct new values.

__weakref__
list of weak references to the object (if defined)

class events.items.preprocessors.**IndexSelector**(*index*)
Selector from a given index on a collection.

The selection will operate on collection[index], so should work for numerical index on *list* and *tuple*, and *dict* for a given key.

index
Index of element to select.

Example

For single use (i.e. outside a pipeline):

```
selector = IndexSelector(0)  # will select first item
value = selector(collection)
```

__call__(*indexable*)
Run selector.

Parameters **indexable** (*indexable collection* or *parsability.ParsableValue*) – Container into which select an element at the given *index*.

Returns Parsed value, i.e. whose `value` attribute is the selection of item indexed as *index* in `indexable` collection.

Return type *parsability.ParsableValue*

__init__(*index*)
Initialization.

Parameters **index** (*int*) – Index on which operate the selection in the collection passed as parameter in *__call__()*.

__weakref__
list of weak references to the object (if defined)

class events.items.preprocessors.**MultiXPathExtractor**(*fields*)
Multiple XPath extractor from a full response.

This is generally the first preprocessor to add to a pipeline.

fields
List of fields for the parsed value, mapped to their related XPath.

Type *OrderedDict* of *str*: *str*

Example

For single use (i.e. outside a pipeline):

```
from collections import OrderedDict

fields = OrderedDict((
    ('odd_paragraphs', '//p[@class="odd"]'),
    ('even_paragraphs', '//p[@class="even"]'),
    ))
extractor = MultiXPathExtractor(fields)
values = extractor(response)

if values.is_parsed():
    for odd, even in values.value:
        # do something
```

__call__(*response*)
Run extractor.

Parameters **response** – Scrapy HTTP response on which run XPaths defined in *fields*.

Returns Parsed value, whose `value` is a *events.utils.collections.IterableNamespace* with fields as declared in *fields*.

Return type *parsability.ParsableValue*

__init__(*fields*)
Initialization.

Parameters **fields** (*OrderedDict* of *str*: *str*) – List of fields for the parsed value, mapped to their related XPath.

__weakref__
list of weak references to the object (if defined)

class events.items.preprocessors.**Object**
Holder for arbitrary attributes.

__repr__()
Return repr(self).

__weakref__
list of weak references to the object (if defined)

class events.items.preprocessors.**PreprocessorPipeline**(**preprocessors*)
Preprocessor pipeline, to use *parsability.ParsableValue* before loading items.

Its purpose is for parsing value before getting any data insertable in an Item, e.g. when extracting data with *XPath*, and use *parsability.ParsableValue* at any stage of this process.

It should be use almost like Item Loader processors, by creating a pipeline of *preprocessors*, and then passing initial value to the pipeline via *__call__()*, in order to have desired result.

preprocessors
List of preprocessors to apply to the initial `value` passed on the pipeline.

Type *list* of *callable*

Example

```
preprocessors = PreprocessorPipeline()
preprocessors.add(Preprocessor1(**params1))
preprocessors.add(Preprocessor2(*params2))
parsed_value = preprocessors(initial_value)
```

__call__(*value*)
Call all preprocessors of the pipeline on an initial value.

Parameters **value** – Initial value passed to the pipeline, on which preprocessors will operate.

Returns Result of `value` passed to all preprocessors.

Return type *parsability.ParsableValue*

__init__(**preprocessors*)
Initialization.

Parameters **preprocessors** (*list* of *callable*) – List of preprocessors to add to the pipeline.

__weakref__
list of weak references to the object (if defined)

add(**preprocessors*)
Add one or more preprocessors.

Parameters **preprocessors** (*list* of *callable*) – List of preprocessors to add to the pipeline.

class events.items.preprocessors.**RegExpExtractor**(*regexp*, *regexp_name*, *fields*)
Regular expression extractor.

Extrat matching groups of a regexp from a given string, and store result in a dedicated `collections.namedtuple`.

regexp
Regular expression used to extract *fields* from a string.

Type `re.RegExp`

regexp_name
Name of *regexp*, used as "reason" of unparsability if any.

Type *str*

fields
Mapping between extraction fields and regexp matching groups.

Type *collections.OrderedDict* of *str*: *str*

namedtuple_class
Data structure to store result of extraction.

Type `collections.namedtuple`

Example

For single use (i.e. outside a pipeline):

```
from collections import OrderedDict
import re

# Parse a sentence like:
#   "Séance du 2017-04-06, au Panthéon, sall 13"
# with matching groups:
# - date = "2017-04-06"
# - loc = "Panthéon, sall 13"
regexp = re.compile((
    r'^'
    r'\s*Séance\s+du\s+'
    r'(?P<date>[-\d]+),\s*'
    r'(au|à)\s+)'
    r'(?P<loc>.*)\s*'
    r'$'
    ))
fields = OrderedDict((
    # (namedtuple key, regexp group)
    ('date', 'date'),
    ('localization', 'loc'),
    ))
extractor = RegExpExtractor(regexp, 'Date and loc. regexp.', fields)
value = extractor("Séance du 2017-04-06, au Panthéon, sall 13")
if value.is_parsed():
    date = value.value.date  # "2017-04-06"
    localization = value.value.localization  # "Panthéon, sall 13"
```

__call__ (*string*)
Extract *fields* of matching *regexp* groups on `string`.

Parameters **string** (`ParsableValue` or *str*) – String from which extract *fields* given the *regexp*.

Returns Parsed value, whose `value` attribute is a *namedtuple_class* with fields containing result of `re.match()` method on *regexp*.

Return type *parsability.ParsableValue*

__init__ (*regexp*, *regexp_name*, *fields*)
Initialization.

Parameters

- **regexp** (`re.RegExp`) – Regular expression used to extract *fields* from a string.
- **regexp_name** (*str*) – Name of *regexp*, used as "reason" of unparsability if any.
- **fields** (*collections.OrderedDict* of *str*: *str*) – Mapping between extraction fields and regexp matching groups.

__weakref__
list of weak references to the object (if defined)

class events.items.preprocessors.**StrSplitter**(*split=None*)
String splitter.

split
Characters where operate the splitting. If set to `None`, split occurs on whitespaces.

Type *str*

Example

For single use (i.e. outside a pipeline):

```
splitter = StrSplitter()  # will operate on blanks
value = splitter(string)
```

__call__(*string*)
Run splitter.

Parameters **string** (*parsability.ParsableValue* or *str*) – String on which operate the splitting.

Returns Parsed value, whose `value` attribute contains the result of applying `str.split()` to `string` with *split* passed as parameter.

Return type *parsability.ParsableValue*

__init__(*split=None*)
Initialization.

Parameters **split** (*str* or `None`) – Characters where operate the splitting. If set to `None`, split occurs on whitespaces.

See also:

https://docs.python.org/3.5/library/stdtypes.html#str.split

__weakref__
list of weak references to the object (if defined)

class events.items.preprocessors.**TagsRemover**(*tags=None*)
HTML tags remover.

tags
List of tags to remove. If empty, *all* tags will be removed.

Type *list* of *str*

Example

For single use (i.e. outside a pipeline):

```
tag_remover = TagsRemover(('p',))
value = tag_remover(html_snippet)
```

__call__(*snippet*)
Run extractor.

Parameters **snippet** (*parsability.ParsableValue* or *str*) – HTML snippet on which remove tags.

Returns Parsed value, i.e. parsable value whose attribute `value` is `snippet` with *tags* removed.

Return type *parsability.ParsableValue*

__init__ (*tags=None*)
Initialization.

Parameters **tags** (*iterable* of *str*, or `None`) – List of tags to remove. If `None` or *empty*, all tags will be removed.

See also:

http://w3lib.readthedocs.io/en/latest/w3lib.html#w3lib.html.remove_tags

__weakref__
list of weak references to the object (if defined)

class events.items.preprocessors.**XPathExtractor** (*xpath*, *method='extract'*)
XPath extractor from a full response.

This is generally the first preprocessor to add to a pipeline.

xpath
XPath to use for extracting value.

Type *str*

method
Method to extract XPath.

Type *str*

Example

For single use (i.e. outside a pipeline):

```
extractor = XPathExtractor('//p[@class="myclass"]', 'extract_first')
value = extractor(response)
```

__call__ (*response*)
Run extractor.

Parameters **response** – Scrapy HTTP response on which run XPath.

Returns Parsed value.

Return type *parsability.ParsableValue*

__init__ (*xpath*, *method='extract'*)
Initialization.

Parameters

- **xpath** (*str*) – XPath to use for extracting value(s).
- **method** (*str*) – Method to extract xpath. Should be `"extract"` (default) or `"extract_first"`, the method will be dynamically called by `getattr()`.

__weakref__
list of weak references to the object (if defined)

events.items.preprocessors.**extract_raw_date** (*parsed_value*)
Extract a *events.utils.date.RawDate* if parsed value is well parsed.

Parameters **parsed_value** (*parsability.ParsableValue*) – Parsed value from which extract a raw date. Its `value` field *should* contains following fields, gettable from a call to `getattr()`:

- `year`,
- `month`,
- `day`.

If a field is missing, its value in returned *RawDate* will be `None`.

Returns Either `parsed_value` if it is not already well parsed, or new parsable value whose `value` attribute is a new *events.utils.date.RawDate*. This *RawDate* could be *incomplete* (see its *is_completed*).

Return type *parsability.ParsableValue*

items.processors

Item Loader processors.

This modules offers some Item Loader processors, inheriting from here defined *Processor*, which implements a generic behavior for bubbling unparsed value from processor to processor.

Processors are:

- *TakeFirstParsed*,
- *init()*,
- *int()*,
- *not_empty()*,
- *remove_tags()*,
- *str2date()*,
- *str2time_duration()*,
- *str_capitalize()*,
- *str_strip()*,
- *str_unquote()*,
- *str_unword()*.

Procssors are used in Item Loader as follows:

```
from scrapy.loader.processors import Identity, MapCompose
from scrapy import Item, Field

from .base import ItemLoader
from .processors import TakeFirstParsed, init, remove_tags, str_strip

class PersonItem(Item):
    name = Field()

class PersonLoader(ItemLoader):
    default_input_processor = Identity()
    default_output_processor = TakeFirstParsed()

    name_in = MapCompose(init, remove_tags, str_strip)
```

class events.items.processors.**Processor**(*callable_*)

Item field processor.

callable

Internal processor. Call to it must always succeed. It should either parse current value, or return `None` in case it could not.

Type *callable*

name

Name of the processor, used as reason to unparsed value. Its value is infered from *callable* it-self, i.e. is value of its `__name__` attribute.

Type *str*

Example

```
def _custom_func(value):
    # process normaly
    # if unprocessable, returns None
    pass

# Make custom func. a processor
custom_func = Processor(_custom_func)
```

__call__(*value*)
Process a parsable value by a processor.

Parameters **value** (*parsability.ParsableValue*) – Parsable value, eventualy already parsed, to be parsed by the processor (i.e. the *callable*).

Returns Parsed value by the processor. If it fails to process value, object returned is set unparsable.

Return type *parsability.ParsableValue*

__init__(*callable_*)
Initialization.

Parameters **callable_** (*callable*) – Internal processor. Call to it must always succeed. It should either parse current value, or return `None` in case it could not.

__weakref__
list of weak references to the object (if defined)

class events.items.processors.**TakeFirstParsed**
"Take first" processor.

It takes first parsed value, or last unparsed. This processor is typicaly set to `default_output_processor` of an Item Loader.

__call__(*values*)
Process a list of values.

Parameters **values** (*iterable* of *parsability.ParsableValue*) – Values from whih take first parsed or last unparsed, loaded by an Item loader.

Returns First fully parsed value of `values`, or last unparsed one.

Return type *parsability.ParsableValue*

__weakref__
list of weak references to the object (if defined)

events.items.processors.**init**(*value*)
First processor to call, wrapping value into a *parsability.ParsableValue*.

Parameters **value** – Value to parse.

Returns Parsable value.

Return type *parsability.ParsableValue*

events.items.processors.**int = <events.items.processors.Processor object>**
Processor transforming a string into an *int*.

Parameters **value** (*parsability.ParsableValue*) – String value to parse into an *int*.

Returns `value` parsed as an *int* if any.

Return type *parsability.ParsableValue*

events.items.processors.**not_empty = <events.items.processors.Processor object>**
Filter for not empty value.

Parameters **string** (parsability.ParseableValue) – String to filter.

Returns None if and only if length of string is null; else string.

Return type *parsability.ParsableValue*

events.items.processors.**remove_tags = <events.items.processors.Processor object>**
HTML tags remover processor.

Parameters **snippet** (*parsability.ParsableValue*) – HTML snippet from which remove all tags.

Returns snippet without HTML tags.

Return type *parsability.ParsableValue*

events.items.processors.**str2date = <events.items.processors.Processor object>**
Processor converting a string into a parsed date.

Parameters **date_repr** (*parsability.ParsableValue*) – Date representation to eval as a true date object.

Returns

date_repr transformed into true datetime.date.

See also:

events.utils.date.parse_date()

Return type *parsability.ParsableValue*

events.items.processors.**str2time_duration = <events.items.processors.Processor object>**
Processor converting a string into a time duration.

Parameters **time_duration_repr** (*parsability.ParsableValue*) – Time duration string representation to parse into an actual time duration object.

Returns

Value of time_duration_repr parsed as a *events.utils.time.TimeDuration*.

See also:

events.utils.time.parse_time_duration()

Return type *parsability.ParsableValue*

events.items.processors.**str_capitalize = <events.items.processors.Processor object>**
String capitalizer processor.

Parameters **string** (*parsability.ParsableValue*) – String to capitalize, i.e. uppering its first char. and lowering others.

Returns string capitalized.

Return type *parsability.ParsableValue*

events.items.processors.**str_strip = <events.items.processors.Processor object>**
String strip processor.

Parameters **string** (*parsability.ParsableValue*) – String whose starting and leading blanks have to be strippped.

Returns string without starting and leading blanks characters.

Return type *parsability.ParsableValue*

events.items.processors.**str_unquote = <events.items.processors.Processor object>**
Unquoter processor, removing starting and leading ' " chars.

Parameters **string** (*parsability.ParsableValue*) – String whose starting and leading quotes (', ") have to be removed.

Returns string without starting and leading simple and double quotes.

Return type *parsability.ParsableValue*

events.items.processors.**str_unword = <events.items.processors.Processor object>**
Processor transforming *MS Word* chars into ASCII ones.

Parameters **string** (*parsability.ParsableValue*) – String containing some *MS Word* chars. to transform into ASCII ones.

Returns

string with *MS Word* copy-pasted chars transformed into ASCII ones.

See also:

events.utils.string.UNWORD_CHARS

Return type *parsability.ParsableValue*

4.2.2 Items

items.person

Person Item to scrap.

class events.items.person.**PersonItem**(**args*, ***kwargs*)
Person infos store.

name = None
Name of the person (*str*).

url = None
URL of its webpage (*str*).

class events.items.person.**PersonLoader**(*item=None*, *selector=None*, *response=None*, *parent=None*, ***context*)
Loader for a *PersonItem*.

default_input_processor = <scrapy.loader.processors.Identity object>
Default input processor.

default_output_processor = <events.items.processors.TakeFirstParsed object>
Default output processor.

name_in = <scrapy.loader.processors.MapCompose object>
List of input processors of *PersonItem.name*.

items.session

Session Item to scrap.

class events.items.session.**SessionItem**(**args*, ***kwargs*)
Event's session infos store.

date = None
(Naïve) Date of the session (`datetime.date`).

localization = None
Localization of the session (*str*).

persons = None
List of persons animating the session (*list* of *person.PersonItem*).

time_duration = None
(Naïve) Time duration of the session (*events.utils.time.TimeDuration*).

title = None
Title of the session (*str*).

class events.items.session.**SessionLoader**(*item=None*, *selector=None*, *response=None*, *parent=None*, ***context*)
Loader for a *SessionItem*.

date_in = <scrapy.loader.processors.MapCompose object>
List of input processors for *SessionItem.date*.

default_input_processor = <scrapy.loader.processors.Identity object>
Default input processor.

default_output_processor = <scrapy.loader.processors.TakeFirst object>
Default output processor.

localization_in = <scrapy.loader.processors.MapCompose object>
List of input processors for *SessionItem.localization*.

persons_out = <scrapy.loader.processors.Identity object>
Output processors for *SessionItem.persons*.

time_duration_in = <scrapy.loader.processors.MapCompose object>
List of input processors for *SessionItem.time_duration*.

title_in = <scrapy.loader.processors.MapCompose object>
List of input processors for *SessionItem.title*.

items.event

Event Item to scrap.

class events.items.event.**EventItem**(**args*, ***kwargs*)
Event infos store.

sessions = None
List of sessions related to the event, if any (*list* of *session.SessionItem*).

title = None
Title of the event (*str*).

type = None
Type of the event (*str*, e.g. `"seminar"`).

url = None
URL of the event, if any (*str*).

class events.items.event.**EventLoader**(*item=None*, *selector=None*, *response=None*, *parent=None*, ***context*)

Loader for a *EventItem*.

default_input_processor = <scrapy.loader.processors.Identity object>
Default input processor.

default_output_processor = <scrapy.loader.processors.TakeFirst object>
Default output processor.

sessions_out = <scrapy.loader.processors.Identity object>
Output processor for *EventItem.sessions*.

title_in = <scrapy.loader.processors.MapCompose object>
Input processors for *EventItem.title*.

4.3 pipelines

4.3.1 pipelines.unparsed

'Unparsed' pipeline.

events.pipelines.unparsed.**LOGGER = <Logger events.pipelines.unparsed (WARNING)>**
Default logger for the spider.

class events.pipelines.unparsed.**UnparsedPipeline**
'Unparsed' pipeline, collecting errors for unparsed values.

__weakref__
list of weak references to the object (if defined)

process_event_item(*event*)
Process an Event item.

Parameters **event** (*events.items.event.EventItem*) – Session item from which collect unparsed values.

Returns First item indicates if `True` that there is no error; it's positioned ar `False` otherwise. Second item is the error(s) of parsability from `event`; it could be a *str* or a *dict*.

Return type *2-uple* of (*bool*, *object*)

process_item(*item*, *spider*)
Pipeline entry point.

process_person_item(*person*)
Process a Person item.

Parameters **person** (*events.items.person.PersonItem*) – Person item from which collect unparsed values.

Returns First item indicates if `True` that there is no error; it's positioned ar `False` otherwise. Second item is the error(s) of parsability from `person`; it could be a *str* or a *dict* of *str*.

Return type *2-uple* of (*bool*, *object*)

process_session_item(*session*)
Process a Session item.

Parameters **session** (*events.items.session.SessionItem*) – Session item from which collect unparsed values.

Returns First item indicates if `True` that there is no error; it's positioned ar `False` otherwise. Second item is the error(s) of parsability from `session`; it could be a *str* or a *dict*.

Return type *2-uple* of (*bool*, *object*)

events.pipelines.unparsed.**unparsable_reason**(*unparsed_value*)
Format unparsed reason and unparasble value.

Parameters **unparsed_value** (*events.items.parsability.ParsableValue*) – Unparsed value.

Returns Formatted unparsed reason and unparsable value.

Return type *str*

4.3.2 pipelines.deparse

'Deparsability' pipeline.

class events.pipelines.deparse.**DeparsePipeline**
'Deparsability' pipeline, extracting parsed value from ParsableValue.

__weakref__
list of weak references to the object (if defined)

process_event_item(*event*)
Process an Event item.

Parameters **event** (*events.items.event.EventItem*) – Session item from which collect unparsed values.

Returns New event item with unparsed fields removed, or `None` if `event` was not parsed.

Return type `None` or *events.items.event.EventItem*

process_item(*item*, *spider*)
Process item, default pipeline entry point.

process_person_item(*person*)
Process a Person item.

Parameters **person** (*events.items.person.PersonItem*) – Person item from which collect unparsed values.

Returns New person item with unparsed fields removed, or `None` if `person` was not parsed.

Return type `None` or *events.items.person.PersonItem*

process_session_item(*session*)
Process a Session item.

Parameters **session** (*events.items.session.SessionItem*) – Session item from which collect unparsed values.

Returns New session item with unparsed fields removed, or `None` if `session` was not parsed.

Return type `None` or *events.items.session.SessionItem*

events.pipelines.deparse.**LOGGER = <Logger events.pipelines.deparse (WARNING)>**
Default logger for the spider.

4.3.3 pipelines.dump

YAML dumper pipeline.

events.pipelines.dump.**LOGGER = <Logger events.pipelines.dump (WARNING)>**
Default logger for the spider.

class events.pipelines.dump.**YAMLDumpPipeline**
YAML dumper pipeline.

__weakref__
list of weak references to the object (if defined)

process_event_item(*event*)
Process an Event item.

Parameters **event** (*events.items.event.EventItem*) – Event item to convert into a *dict*.

Returns Fields and values from `event`.

Return type *dict*

process_item(*item*, *spider*)
Process item, pipeline default entry point.

process_person_item(*person*)
Process a Person item.

Parameters **person** (*events.items.person.PersonItem*) – Person item to convert into a *dict*.

Returns Fields and values from `person`.

Return type *dict*

process_session_item(*session*)
Process a Session item.

Parameters **session** (*events.items.session.SessionItem*) – Session item to convert into a *dict*.

Returns Fields and values from `session`.

Return type *dict*

4.3.4 pipelines.utils

Utilities for Item pipelines.

events.pipelines.utils.**extract_parsed_value**(*item*, *attr_name*)
Extract parsed value if any.

Note: This is a pure function.

Parameters

- **item** (`scrapy.Item` or *events.items.parsability.ParsableValue*) – Item from which extract parsed value, if any.
- **attr_name** (*str*) – Attribute name attendue to belong to `item`.

Returns First item is state about parsed value: `True` if value well extracted, `False` otherwise. Second item is either the extracted value if first item is `True`, or else an error message.

Return type *2-uple* of (*bool*, *object*)

4.4 utils

4.4.1 utils.collections

Custom collections.

class events.utils.collections.**IterableNamespace**(*attrs*)
Namespace which is iterable on its own iterable attributes.

Its main purpose is to purpose iteration with `zip()` over all of its attributes, which muse be also iterables.

attr_names
List of attributes names, ordered.

Type *list* of *str*

Example

A basic usage could be as follows:

```
from collections import OrderedDict

fields = OrderedDict((
    ('name1', [v11, v12, v13]),
    ('name2', [v21, v22, v23]),
    ))
ins = IterableNamespace(fields)

for name1, name2 in ins:
    # for the first iteration:
    #   name1 = v11; name2 = v21
    # do something
    pass
```

__init__(*attrs*)
Initialization.

Parameters **attrs** (*collections.OrderedDict*) – *Ordered* map of attributes values mapped by their names.

__iter__()
Iteration over attributes, through `zip()`.

property attr_names
Get an immutable copy of attributes names, ordered.

4.4.2 utils.date

Date related utilities.

events.utils.date.**MONTHS_TO_STRINGS = {1: ['1', 'janvier', 'janv'], 2: ['2', 'fevrier',**
Months numbers mapped to their potential representations (*dict* of *int*: *list* of *str*).

events.utils.date.**MONTH_FROM_STRINGS = {'1': 1, '10': 10, '11': 11, '12': 12, '2': 2, '3**
Months numbers mapped *from* their potential representations (*dict* of *str*: *int*).

class events.utils.date.**RawDate**(*year=None*, *month=None*, *day=None*)
Raw and naïve date with fields for day/month/year for temporay storage.

Its fields are not typed, so that it could be anything (e.g. for *month*: `12`, `"December"`, etc.), and used indifferently at several different steps of parsing.

It provides a convenient method *is_completed()* to tell if all of its fields are provided (i.e. not `None`).

Note: Immutable structure!

Note that a *RawDate* is immutable by nature: once its fields are set through *__init__()*, they can not be altered, only accessed.

---

year
Field for storing date's year.

Type no type

month
Field for storing date's month.

Type no type

day
Field for storing date's day.

Type no type

__init__(*year=None*, *month=None*, *day=None*)
Initialization.

Parameters

- **year** (*no type*) – Year of the date.
- **month** (*no type*) – Month of the date.
- **day** (*no type*) – Day of the date.

__repr__()
Return repr(self).

__weakref__
list of weak references to the object (if defined)

is_completed()
Tell if a date is completed or not.

Returns `True` if all fields are completed (i.e. not `None`), `False` otherwise.

Return type *bool*

events.utils.date.**month_number**(*month_repr*)
Get month number from its representation.

Parameters **month_repr** (*str*) – Month representation from which infer month number (in a Gregorian calendar).

Note: *ASCII* and lower characters only!

Currently, months are infered from:

- their number (in the Gregorian Calendar);
- their french representation without 1) accent, 2) capital.

Returns

Either month number if inference was successful, or `month_repr`.

See also:

MONTH_FROM_STRINGS conversion mapping.

Return type *int* or *str*

Example

```
from unidecode import unidecode

month = "Décembre"
month = unidecode(month.lower())
month = month_number(month)
```

events.utils.date.**parse_date**(*parsed_date*)
Parse a raw date field into a `datetime.Date`.

Parameters **parsed_date** (*RawDate*) – Parsed date, as raw parsed by spider.

Returns Fully parsed date object, or `None` if unparsable. Unparsability could come from uninferrable month by *month_number()*, or by invalid values of day/month/year (e.g. 35 December of -1 year).

Return type `datetime.date` or `None`

Example

```
date = parse_date(RawDate("2017", "Décembre", "31"))
```

events.utils.date.**year_from_month**(*month_number*)
Get current academic year depending on month, assuming current year.

Note: This function assumes that academic year starts at September (included) of year YYYY, and finishes at August (included) of year YYYY + 1.

Parameters **month_number** (*int*) – Month number from which get year.

Returns Year of the month.

Return type *int*

4.4.3 utils.string

String related utilities.

events.utils.string.**UNWORD_CHARS = {'’': "'", '“': '"', '”': '"'}**
Translation map for special characters, typically inserted by copy/pasting from *MS Word* (*dict* of *str*:*str*).

events.utils.string.**UNWORD_TABLE = {8217: "'", 8220: '"', 8221: '"'}**
Translation table for special characters, typically inserted by copy/pasting from *MS Word*, constructed from *UNWORD_CHARS* and usable by `str.translate()`.

events.utils.string.**unquote_str**(*string*)
Remove starting and leading simple and double quotes (`'`, `"`) of a string.

Parameters **string** (*str*) – String to unquote.

Returns Unquoted string.

Return type *str*

Example

```
string = '"L\'inconscient"'
string = unquote_str(string)
string == "L'inconscient"
```

events.utils.string.**unword_str**(*string*)
Translate special characters from pasting from Word into regular ones.

Parameters **string** (*str*) – String to translate.

Returns

Translated string, with special characters copy-pasted from *MS Word* translated into ASCII ones.

See also:

UNWORD_CHARS for translation mapping.

Return type *str*

Example

```
string = "“L’inconscient”"
string = unword_str(string)
string == "\"L'inconscient\""
```

4.4.4 utils.time

Time related utilities.

events.utils.time.**DURATION_FROM_STRING_RE = re.compile('^(?P<starting_hour>[0-9][0-9]?)**
Duration regexp for extraction from a string representation (re.RegExp).

class events.utils.time.**TimeDuration**(*start*, *end=None*)
Intervention duration, with a starting and optional ending time.

Note: Immutable structure!

Note that a *TimeDuration* is immutable by nature: once its fields are set through *__init__()*, they can not be altered, only accessed.

start
Time where duration starts.

Type datetime.time

end
Time where duration ends.

Type datetime.time

__init__(*start*, *end=None*)
Initialization.

Parameters

- **start** (datetime.time) – Time where duration starts.
- **end** (datetime.time) – Time where duration ends.

__repr__()
Return repr(self).

__weakref__
list of weak references to the object (if defined)

has_end()
Tell if the duration has an end.

Returns `True` if duration has an *end* set, `False` otherwise.

Return type *bool*

`events.utils.time.`**`parse_time_duration`**(*duration_repr*)

Get value object of intervention duration from its string representation.

Parameters **`duration_repr`** (*str*) – Time duration representation from which infer its value.

Returns

Either value objet if parsing was successful, `None` otherwise.

See also:

DURATION_FROM_STRING_RE for parsing regexp.

Return type *TimeDuration* or `None`

Example

```
duration = "17h-19.30h"
duration = parse_time_duration(duration)
duration.start.hour == 17
duration.end.minute == 30
```

CHAPTER

FIVE

CURRENT STATE OF DATABASES

5.1 *MySQL* schema

Field	Type	Key
IDManifestation	int(5)	PRIMARY_KEY
IDLieu	int(5)	
Type	varchar(50)	
Connection	char(1)	
intitule	varchar(250)	INDEX
soustitre	varchar(250)	
precision_manifestation	text	
organisation	text	
presentation	text	
date_de_debut	date	
date_de_fin	date	
frequence	varchar(250)	
url	varchar(250)	
horaires	varchar(25)	
IDResponsable1	int(11)	
IDResponsable2	int(11)	
IDResponsable3	int(11)	
acces	varchar(25)	
etage	varchar(250)	
batiment	varchar(250)	
pavillon	varchar(250)	
salle	varchar(250)	
escalier	varchar(250)	
couloir	varchar(250)	
region	varchar(100)	INDEX??
chemin_doc_attache	varchar(250)	

```
create table Manifestation (
    `IDManifestation` int(5) NOT NULL,
    `IDLieu` int(5) DEFAULT NULL,
    `Type` varchar(50) COLLATE utf8_bin DEFAULT NULL,
    `Connection` char(1) COLLATE utf8_bin DEFAULT NULL,
    `intitule` varchar(250) COLLATE utf8_bin DEFAULT NULL,
    `soustitre` varchar(250) COLLATE utf8_bin DEFAULT NULL,
    `precision_manifestation` text COLLATE utf8_bin,
    `organisation` text COLLATE utf8_bin,
    `presentation` text COLLATE utf8_bin,
    `date_de_debut` date DEFAULT NULL,
    `date_de_fin` date DEFAULT NULL,
```

(continues on next page)

(continued from previous page)

```
    `frequence` varchar(250) COLLATE utf8_bin DEFAULT NULL,
    `url` varchar(250) COLLATE utf8_bin DEFAULT NULL,
    `horaires` varchar(25) COLLATE utf8_bin DEFAULT NULL,
    `IDResponsable1` int(11) DEFAULT NULL,
    `IDResponsable2` int(11) DEFAULT NULL,
    `IDResponsable3` int(11) DEFAULT NULL,
    `acces` varchar(25) COLLATE utf8_bin DEFAULT NULL,
    `etage` varchar(250) COLLATE utf8_bin DEFAULT NULL,
    `batiment` varchar(250) COLLATE utf8_bin DEFAULT NULL,
    `pavillon` varchar(250) COLLATE utf8_bin DEFAULT NULL,
    `salle` varchar(250) COLLATE utf8_bin DEFAULT NULL,
    `escalier` varchar(250) COLLATE utf8_bin DEFAULT NULL,
    `couloir` varchar(250) COLLATE utf8_bin DEFAULT NULL,
    `region` varchar(100) COLLATE utf8_bin DEFAULT NULL,
    `chemin_doc_attache` varchar(250) COLLATE utf8_bin DEFAULT NULL,
    PRIMARY KEY (`IDManifestation`),
    KEY `RegionIndex` (`region`),
    KEY `IntituleIndex` (`intitule`)
);
```

5.2 *FileMaker Pro*

CHAPTER

SIX

OTHER PAGES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX

Non-alphabetical

A

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

X

Y