# raspador Documentation

***Release 0.2.2***

**Fernando Macedo**

September 21, 2015

Contents

CHAPTER 1

# Install

## 1.1 Package managers

You can install using pip or easy_install.

PIP:

```
pip install raspador
```

Easy install:

```
easy_install raspador
```

## 1.2 From source

Download and install from source:

```
git clone https://github.com/fgmacedo/raspador.git
cd raspador
python setup.py install
```

CHAPTER 2

# raspador

Library to extract data from semi-structured text documents.

It's best suited for data-processing in files that do not have a formal structure and are in plain text (or that are easy to convert).

## 2.1 Parser

**class** raspador.parser.**ParserMetaclass**(*name*, *bases*, *attrs*)
Collect data-extractors into a field collection and injects ParserMixin.

**class** raspador.parser.**ParserMixin**
A mixin that holds all base parser implementation.

**default_item_class**
alias of Dictionary

**process_item**(*item*)
Allows final modifications at the object being returned

## 2.2 Fields

Fields define how and what data will be extracted. The parser does not expect the fields explicitly inherit from *BaseField*, the minimum expected is that a field has at least a method *parse_block*.

The fields in this file are based on regular expressions and provide conversion for primitive types in Python.

**class** raspador.fields.**BRFloatField**(*search*, *thousand_separator=None*, *decimal_separator=None*, ***kwargs*)
Removes thousand separator and converts to float (Brazilian format).

Deprecated since version 0.2.2: Use *FloatField* instead.

**default_decimal_separator = ','**

**default_thousand_separator = '.'**

**class** raspador.fields.**BaseField**(*search=None*, *default=None*, *is_list=False*, *input_processor=None*, *groups=[]*)
Contains processing logic to extract data using regular expressions, and provide utility methods that can be overridden for custom data processing.

Default behavior can be adjusted by parameters:

search

Regular expression that must specify a group of capture. Use parentheses for capturing:

```
>>> s = "02/01/2013 10:21:51           COO:022734"
>>> field = BaseField(search=r'COO:(\d+)')
>>> field.parse_block(s)
'022734'
```

The *search* parameter is the only by position and hence its name can be omitted:

```
>>> s = "02/01/2013 10:21:51           COO:022734"
>>> field = BaseField(r'COO:(\d+)')
>>> field.parse_block(s)
'022734'
```

input_processor

Receives a function to handle the captured value before being returned by the field.

```
>>> s = "02/01/2013 10:21:51           COO:022734"
>>> def double(value):
...     return int(value) * 2
...
>>> field = BaseField(r'COO:(\d+)', input_processor=double)
>>> field.parse_block(s)  # 45468 = 2 x 22734
45468
```

groups

Specify which numbered capturing groups do you want do process in.

You can enter a integer number, as the group index:

```
>>> s = "Contador de Reduções Z:                    1246"
>>> regex = r'Contador de Reduç(ão|ões) Z:\s*(\d+)'
>>> field = BaseField(regex, groups=1, input_processor=int)
>>> field.parse_block(s)
1246
```

Or a list of integers:

```
>>> s = "Data do movimento: 02/01/2013 10:21:51"
>>> regex = r'^Data .*(movimento|cupom): (\d+)/(\d+)/(\d+)'
>>> c = BaseField(regex, groups=[1, 2, 3])
>>> c.parse_block(s)
['02', '01', '2013']
```

---

**Note:** If you do not need the group to capture its match, you can optimize the regular expression putting an *?:* after the opening parenthesis:

```
>>> s = "Contador de Reduções Z:                    1246"
>>> field = BaseField(r'Contador de Reduç(?:ão|ões) Z:\s*(\d+)')
>>> field.parse_block(s)
```

'1246'

---

default

If assigned, the `Parser` will query this default if no value was returned by the field.

is_list

When specified, returns the value as a list:

```
>>> s = "02/01/2013 10:21:51           COO:022734"
>>> field = BaseField(r'COO:(\d+)', is_list=True)
>>> field.parse_block(s)
['022734']
```

By convention, when a field returns a list, the `Parser` accumulates values

returned by the field.

**assign_class**(*cls*, *name*)

**assign_parser**(*parser*)
Receives a weak reference of `Parser`

**parse_block**(*block*)

**search**

**setup**()
Hook to special setup required on child classes

**to_python**(*value*)
Converts parsed data to native python type.

**class** raspador.fields.**BooleanField**(*search=None*, *default=None*, *is_list=False*, *input_processor=None*, *groups=[]*)
Returns true if the block is matched by Regex, and is at least some value is captured.

**setup**()

**to_python**(*value*)

**class** raspador.fields.**DateField**(*search=None*, *format_string=None*, ***kwargs*)
Field that holds data in date format, represented in Python by datetine.date.

http://docs.python.org/library/datetime.html

**convertion_function**(*date*)

**default_format_string = '%d/%m/%Y'**

**to_python**(*value*)

**class** raspador.fields.**DateTimeField**(*search=None*, *format_string=None*, ***kwargs*)
Field that holds data in hour/date format, represented in Python by datetine.datetime.

http://docs.python.org/library/datetime.html

**convertion_function**(*date*)

**default_format_string = '%d/%m/%Y %H:%M:%S'**

**class** raspador.fields.**FloatField**(*search*, *thousand_separator=None*, *decimal_separator=None*, ***kwargs*)
Sanitizes captured value according to thousand and decimal separators and converts to float.

**default_decimal_separator = '.'**

**default_thousand_separator = ','**

**to_python**(*value*)

**class** raspador.fields.**IntegerField**(*search=None*, *default=None*, *is_list=False*, *input_processor=None*, *groups=[]*)

**to_python**(*value*)

**class** raspador.fields.**StringField**(*search=None*, *default=None*, *is_list=False*, *input_processor=None*, *groups=[]*)

**to_python**(*value*)

## 2.3 Item

**class** raspador.item.**Dictionary**(**args*, ***kwds*)
Dictionary that exposes keys as properties for easy read access.

CHAPTER 3

# Community updates

If you'd like to stay up to date on the development of Raspador, there are several options:

## 3.1 GitHub

The best way to track the development of Raspador is through the GitHub repo.

## 3.2 Twitter

I often tweet about new features and releases of Raspador.

Follow @fgmacedo for updates.

CHAPTER 4

# Release history

0.2.3

- Linux line endings (thanks Jayson Reis - jaysonsantos)

## 4.1 0.2.2 (2013-10-30)

- More tests translated to `en`.
- More useful log messages.

**API Changes:**

- `BaseField._setup` renamed to `BaseField.setup`.
- `FloatField` has new parameters and defaults: `thousand_separator` and `decimal_separator`.
- `BRFloatField` is now deprecated in favor of `FloatField` parameters.

Looking for specific information? Try the genindex or modindex.

## r

## A



## B



## C



## D



## F



## I



## P



## R



## S



## T