BibtexParser Documentation

Release 1.0

F. Boulogne

Feb 16, 2018

Contents

Author François Boulogne and other contributors

Devel github.com project

Mirror git.sciunto.org

Bugs github.com

Generated Feb 16, 2018

Version 1.0

BibtexParser is a python library to parse bibtex files. The code relies on pyparsing and is tested with unittests.

If you use BibtexParser for your project, feel free to send me an email. I would be happy to hear that and to mention your project in the documentation.

Contents:

CHAPTER 1

How to install and test?

1.1 How to install?

1.1.1 Requirements

- python **2.7** or python **3.3** or newer
- pyparsing **2.0.3** or newer

1.1.2 Package manager (recommended for those OS users)

- Archlinux
- Debian

1.1.3 pip (recommended to other users)

To install with pip:

```
pip install bibtexparser
```

1.1.4 Manual installation (recommended for packagers)

Download the archive on Pypi.

```
python setup.py install
```

1.2 How to run the test suite?

This paragraph briefly describes how to run the test suite. This is useful for contributors, for packagers but also for users who wants to check their environment.

1.2.1 Virtualenv

You can make a virtualenv. I like pew for that because the API is easier.

The first time, you need to make a virtualenv

```
pew mkproject bibtexparser
pip install -r requirements.txt
python setup.py install
nosetest
```

If you already have a virtualenv, you can use workon

```
pew workon bibtexparser
```

1.2.2 Tox

The advantage of Tox is that you can build and test the code against several versions of python. Of course, you need tox to be installed on your system. The configuration file is tox.ini, in the root of the project. There, you can change the python versions.

```
tox # and nothing more :)
```

CHAPTER 2

Tutorial

2.1 Step 0: Vocabulary

- An **entry** designates for example *@book{...}*, *@article{...}*, etc.
- A **comment** is written as *@comment{...}*.
- A **preamble** is a *@preamble{...}* block.
- A **string** is *@string{...}*.

In an entry, you can find

- an **entry type** like *article*, *book*, etc.
- **entry keys** or **keys** such as *author*, *title*, *year*...
- and also **records**, which designates the values of those keys.

2.2 Step 1: Prepare a BibTeX file

First, we prepare a BibTeX sample file. This is just for the purpose of illustration:

```
bibtex = """@ARTICLE{Cesar2013,
  author = {Jean César},
  title = {An amazing title},
  year = {2013},
  month = jan,
  volume = {12},
  pages = {12--23},
  journal = {Nice Journal},
  abstract = {This is an abstract. This line should be long enough to test
     multilines...},
  comments = {A comment},
  keywords = {keyword1, keyword2}
```

```
}
"""

with open('bibtex.bib', 'w') as bibfile:
    bibfile.write(bibtex)
```

2.3 Step 2: Parse it!

2.3.1 Simplest call

OK. Everything is in place. Let's parse the BibTeX file.

```
import bibtexparser

with open('bibtex.bib') as bibtex_file:
    bib_database = bibtexparser.load(bibtex_file)

print(bib_database.entries)
```

It prints a list of dictionaries for reference entries, for example books, articles:

```
[{'journal': 'Nice Journal',
  'comments': 'A comment',
  'pages': '12--23',
  'month': 'jan',
  'abstract': 'This is an abstract. This line should be long enough to␣
↪test\nmultilines...',
  'title': 'An amazing title',
  'year': '2013',
  'volume': '12',
  'ID': 'Cesar2013',
  'author': 'Jean César',
  'keyword': 'keyword1, keyword2',
  'ENTRYTYPE': 'article'}]
```

Note that, by convention, uppercase keys (ID, ENTRYTYPE) are data generated by the parser, while lowercase keys come from the original bibtex file.

You can also print comments, preambles and string:

```
print(bib_database.comments)
print(bib_database.preambles)
print(bib_database.strings)
```

2.3.2 Parse a string

If for some reason, you prefer to parse a string, that's also possible:

```
import bibtexparser

with open('bibtex.bib') as bibtex_file:
    bibtex_str = bibtex_file.read()

bib_database = bibtexparser.loads(bibtex_str)
```

2.3.3 Tune parser's options

In the previous snippet, several default options are used. You can tweak them as you wish.

```
import bibtexparser
from bibtexparser.bparser import BibTexParser

    parser = BibTexParser()
    parser.ignore_nonstandard_types = False
    parser.homogenise_fields = False
    parser.common_strings = False

    bib_database = bibtexparser.loads(bibtex_str, parser)
```

2.4 Step 3: Export

Once you worked on your parsed database, you may want to export the result. This library provides some functions to help on that. However, you can write your own functions if you have specific requirements.

2.4.1 Create a BibTeX file or string

The bibliographic data can be converted back into a string :

```
import bibtexparser

bibtex_str = bibtexparser.dumps(bib_database)
```

or a BibTeX file like this:

```
import bibtexparser

with open('bibtex.bib', 'w') as bibtex_file:
    bibtexparser.dump(bibtex_database, bibtex_file)
```

2.4.2 Call the writer

In the first section we prepared a BibTeX sample file, we can prepare the same file using pure python and the `BibTexWriter` class.

```
from bibtexparser.bwriter import BibTexWriter
from bibtexparser.bibdatabase import BibDatabase

db = BibDatabase()
db.entries = [
    {'journal': 'Nice Journal',
     'comments': 'A comment',
     'pages': '12--23',
     'month': 'jan',
     'abstract': 'This is an abstract. This line should be long enough to␣
↪test\nmultilines...',
```

```
     'title': 'An amazing title',
     'year': '2013',
     'volume': '12',
     'ID': 'Cesar2013',
     'author': 'Jean César',
     'keyword': 'keyword1, keyword2',
     'ENTRYTYPE': 'article'}]

writer = BibTexWriter()
with open('bibtex.bib', 'w') as bibfile:
    bibfile.write(writer.write(db))
```

This code generates the following file:

```
@article{Cesar2013,
 abstract = {This is an abstract. This line should be long enough to test
multilines...},
 author = {Jean César},
 comments = {A comment},
 journal = {Nice Journal},
 keyword = {keyword1, keyword2},
 month = {jan},
 pages = {12--23},
 title = {An amazing title},
 volume = {12},
 year = {2013}
}
```

The writer also has several flags that can be enabled to customize the output file. For example we can use `indent` and `comma_first` to customize the previous entry, first the code:

```
from bibtexparser.bwriter import BibTexWriter
from bibtexparser.bibdatabase import BibDatabase

db = BibDatabase()
db.entries = [
    {'journal': 'Nice Journal',
     'comments': 'A comment',
     'pages': '12--23',
     'month': 'jan',
     'abstract': 'This is an abstract. This line should be long enough to␣
↪test\nmultilines...',
     'title': 'An amazing title',
     'year': '2013',
     'volume': '12',
     'ID': 'Cesar2013',
     'author': 'Jean César',
     'keyword': 'keyword1, keyword2',
     'ENTRYTYPE': 'article'}]

writer = BibTexWriter()
writer.indent = '    '     # indent entries with 4 spaces instead of one
writer.comma_first = True  # place the comma at the beginning of the line
with open('bibtex.bib', 'w') as bibfile:
    bibfile.write(writer.write(db))
```

This code results in the following, customized, file:

```
@article{Cesar2013
,     abstract = {This is an abstract. This line should be long enough to test
multilines...}
,     author = {Jean César}
,     comments = {A comment}
,     journal = {Nice Journal}
,     keyword = {keyword1, keyword2}
,     month = {jan}
,     pages = {12--23}
,     title = {An amazing title}
,     volume = {12}
,     year = {2013}
}
```

Flags to the writer object can modify not only how an entry is printed but how several BibTeX entries are sorted and separated. See the bibtexparser_api for the full list of flags.

2.5 Step 4: Add salt and pepper

In this section, we discuss about some customizations and details.

2.5.1 Customizations

By default, the parser does not alter the content of each field and keeps it as a simple string. There are many cases where this is not desired. For example, instead of a string with a multiple of authors, it could be parsed as a list.

To modify field values during parsing, a callback function can be supplied to the parser which can be used to modify BibTeX entries. The library includes several functions which may be used. Alternatively, you can read them to create your own functions.

```
import bibtexparser
from bibtexparser.bparser import BibTexParser
from bibtexparser.customization import *

# Let's define a function to customize our entries.
# It takes a record and return this record.
def customizations(record):
    """Use some functions delivered by the library

    :param record: a record
    :returns: -- customized record
    """
    record = type(record)
    record = author(record)
    record = editor(record)
    record = journal(record)
    record = keyword(record)
    record = link(record)
    record = page_double_hyphen(record)
    record = doi(record)
    return record

with open('bibtex.bib') as bibtex_file:
    parser = BibTexParser()
```

```
    parser.customization = customizations
    bib_database = bibtexparser.load(bibtex_file, parser=parser)
    print(bib_database.entries)
```

If you think that you have a customization which could be useful to others, please share with us!

2.5.2 Accents and weird characters

Your bibtex may contain accents and specific characters. They are sometimes coded like this `\'{e}` but this is not the correct way, `{\'e}` is preferred. Moreover, you may want to manipulate `é`. There is different situations:

- Case 1: you plan to use this library to work with latex and you assume that the original bibtex is clean. You have nothing to do.
- Case 2: you plan to use this library to work with latex but your bibtex is not really clean.

```
import bibtexparser
from bibtexparser.bparser import BibTexParser
from bibtexparser.customization import homogenize_latex_encoding

with open('bibtex.bib') as bibtex_file:
    parser = BibTexParser()
    parser.customization = homogenize_latex_encoding
    bib_database = bibtexparser.load(bibtex_file, parser=parser)
    print(bib_database.entries)
```

- Case 3: you plan to use this library to work with something different and your bibtex is not really clean. Then, you probably want to use unicode.

```
import bibtexparser
from bibtexparser.bparser import BibTexParser
from bibtexparser.customization import convert_to_unicode

with open('bibtex.bib') as bibtex_file:
    parser = BibTexParser()
    parser.customization = convert_to_unicode
    bib_database = bibtexparser.load(bibtex_file, parser=parser)
    print(bib_database.entries)
```

Note: If you want to mix different customization functions, you can write your own function.

2.5.3 Using bibtex strings

Warning: support for bibtex strings representation is still an experimental feature; the way strings are represented is likely to change in future releases.

Bibtex strings and string expressions are expanded by default into the value they represent. This behavior is controlled by the `interpolate_string` argument of the BibTexParser. It defaults to `True` but can be set to `False`, in which case bibtex strings and string expressions from input files are represented with the `bibdatabase.BibDataString` and `bibdatabase.BibDataStringExpression` from the `bibdatabase` module. Both classes retain the intrinsic structure of the string or expression so that they can be written to a new file, the same way.

Each instance provides a `get_value()` method to interpolate the string or expression and the module also provide an `bibdatabase.as_text()` helper to expand a string or an expression when needed.

Using the code would yield the following output.

```
from bibtexparser.bparser import BibTexParser
from bibtexparser.bibdatabase import as_text


bibtex = """@STRING{ jean = "Jean"}

@ARTICLE{Cesar2013,
  author = jean # { César},
  title = {An amazing title},
  year = {2013},
  month = jan,
  volume = {12},
  pages = {12--23},
  journal = {Nice Journal},
}
"""

bp = BibTexParser(interpolate_strings=False)
bib_database = bp.parse(bibtex)
bib_database.entries[0]
as_text(bd.entries[0]['author'])
```

```
{'ENTRYTYPE': 'article',
 'ID': 'Cesar2013',
 'author': BibDataStringExpression([BibDataString('jean'), ' César']),
 'journal': 'Nice Journal',
 'month': BibDataStringExpression([BibDataString('jan')]),
 'pages': '12--23',
 'title': 'An amazing title',
 }
'Jean César'
```

Contents

CHAPTER 3

bibtexparser: API

3.1 `bibtexparser` — Parsing and writing BibTeX files

BibTeX is a bibliographic data file format.

The *bibtexparser* module can parse BibTeX files and write them. The API is similar to the `json` module. The parsed data is returned as a simple `BibDatabase` object with the main attribute being `entries` representing bibliographic sources such as books and journal articles.

The following functions provide a quick and basic way to manipulate a BibTeX file. More advanced features are also available in this module.

Parsing a file is as simple as:

```
import bibtexparser
with open('bibtex.bib') as bibtex_file:
   bibtex_database = bibtexparser.load(bibtex_file)
```

And writing:

```
import bibtexparser
with open('bibtex.bib', 'w') as bibtex_file:
    bibtexparser.dump(bibtex_database, bibtex_file)
```

bibtexparser.**load**(*bibtex_file*, *parser=None*)

Load `BibDatabase` object from a file

Parameters

- **bibtex_file** (*file*) – input file to be parsed
- **parser** (`BibTexParser`) – custom parser to use (optional)

Returns bibliographic database object

Return type *BibDatabase*

Example:

```
import bibtexparser
with open('bibtex.bib') as bibtex_file:
   bibtex_database = bibtexparser.load(bibtex_file)
```

bibtexparser.**loads**(*bibtex_str*, *parser=None*)
Load BibDatabase object from a string

Parameters

- **bibtex_str** (*str or unicode*) – input BibTeX string to be parsed
- **parser** (BibTexParser) – custom parser to use (optional)

Returns bibliographic database object

Return type *BibDatabase*

bibtexparser.**dumps**(*bib_database*, *writer=None*)
Dump BibDatabase object to a BibTeX string

Parameters

- **bib_database** (BibDatabase) – bibliographic database object
- **writer** (BibTexWriter) – custom writer to use (optional) (not yet implemented)

Returns BibTeX string

Return type unicode

bibtexparser.**dump**(*bib_database*, *bibtex_file*, *writer=None*)
Dump BibDatabase object as a BibTeX text file

Parameters

- **bib_database** (BibDatabase) – bibliographic database object
- **bibtex_file** (*file*) – file to write to
- **writer** (BibTexWriter) – custom writer to use (optional) (not yet implemented)

Example:

```
import bibtexparser
with open('bibtex.bib', 'w') as bibtex_file:
    bibtexparser.dump(bibtex_database, bibtex_file)
```

3.2 bibtexparser.bibdatabase — The bibliographic database object

class bibdatabase.**BibDatabase**
Bibliographic database object that follows the data structure of a BibTeX file.

comments = None
List of BibTeX comment (*@comment{...}*) blocks.

entries = None
List of BibTeX entries, for example *@book{...}*, *@article{...}*, etc. Each entry is a simple dict with BibTeX field-value pairs, for example *'author': 'Bird, R.B. and Armstrong, R.C. and Hassager, O.'* Each entry will always have the following dict keys (in addition to other BibTeX fields):

- *ID* (BibTeX key)
- *ENTRYTYPE* (entry type in lowercase, e.g. *book*, *article* etc.)

entries_dict
Return a dictionary of BibTeX entries. The dict key is the BibTeX entry key

preambles = None
List of BibTeX preamble (*@preamble{...}*) blocks.

strings = None
OrderedDict of BibTeX string definitions (*@string{...}*). In order of definition.

3.3 bibtexparser.bparser — Tune the default parser

class bparser.**BibTexParser**(*data=None*, *customization=None*, *ignore_nonstandard_types=True*, *homogenize_fields=False*, *interpolate_strings=True*, *common_strings=False*)
A parser for reading BibTeX bibliographic data files.

Example:

```
from bibtexparser.bparser import BibTexParser

bibtex_str = ...

parser = BibTexParser()
parser.ignore_nonstandard_types = False
parser.homogenize_fields = False
parser.common_strings = False
bib_database = bibtexparser.loads(bibtex_str, parser)
```

Parameters

- **customization** – function or None (default) Customization to apply to parsed entries.
- **ignore_nonstandard_types** – bool (default True) If True ignores non-standard bibtex entry types.
- **homogenize_fields** – bool (default False) Common field name replacements (as set in alt_dict attribute).
- **interpolate_strings** – bool (default True) If True, replace bibtex string by their value, else uses BibDataString objects.
- **common_strings** – book (default False) Include common string definitions (e.g. month abbreviations) to the bibtex file.

common_strings = None
Load common strings such as months abbreviation Default: *False*.

customization = None
Callback function to process BibTeX entries after parsing, for example to create a list from a string with multiple values. By default all BibTeX values are treated as simple strings. Default: *None*.

homogenize_fields = None
Sanitize BibTeX field names, for example change *url* to *link* etc. Field names are always converted to lowercase names. Default: *False*.

ignore_nonstandard_types = None
Ignore non-standard BibTeX types (*book*, *article*, etc). Default: *True*.

interpolate_strings = None
Interpolate Bibtex Strings or keep the structure

parse(*bibtex_str*, *partial=False*)
Parse a BibTeX string into an object

Parameters

- **bibtex_str** – BibTeX string
- **partial** – If True, print errors only on parsing failures.

Type str or unicode

If False, an exception is raised. :type: boolean :return: bibliographic database :rtype: BibDatabase

parse_file(*file*, *partial=False*)
Parse a BibTeX file into an object

Parameters

- **file** – BibTeX file or file-like object
- **partial** – If True, print errors only on parsing failures.

Type file

If False, an exception is raised. :type: boolean :return: bibliographic database :rtype: BibDatabase

3.4 bibtexparser.customization — Functions to customize records

A set of functions useful for customizing bibtex fields. You can find inspiration from these functions to design yours. Each of them takes a record and return the modified record.

customization.**splitname**(*name*, *strict_mode=True*)
Break a name into its constituent parts: First, von, Last, and Jr.

Parameters

- **name** (*string*) – a string containing a single name
- **strict_mode** (*Boolean*) – whether to use strict mode

Returns dictionary of constituent parts

Raises ***customization.InvalidName*** – If an invalid name is given and `strict_mode = True`.

In BibTeX, a name can be represented in any of three forms:

- First von Last
- von Last, First
- von Last, Jr, First

This function attempts to split a given name into its four parts. The returned dictionary has keys of `first`, `last`, `von` and `jr`. Each value is a list of the words making up that part; this may be an empty list. If the input has no non-whitespace characters, a blank dictionary is returned.

It is capable of detecting some errors with the input name. If the `strict_mode` parameter is `True`, which is the default, this results in a *customization.InvalidName* exception being raised. If it is `False`, the function continues, working around the error as best it can. The errors that can be detected are listed below along with the handling for non-strict mode:

- Name finishes with a trailing comma: delete the comma
- Too many parts (e.g., von Last, Jr, First, Error): merge extra parts into First
- Unterminated opening brace: add closing brace to end of input
- Unmatched closing brace: add opening brace at start of word

customization.**getnames**(*names*)

Convert people names as surname, firstnames or surname, initials.

Parameters **names** (*list*) – a list of names

Returns list – Correctly formated names

This function is known to be too simple to handle properly the complex rules. We would like to enhance this in forthcoming releases.

customization.**author**(*record*)

Split author field into a list of "Name, Surname".

Parameters **record** (*dict*) – the record.

Returns dict – the modified record.

customization.**editor**(*record*)

Turn the editor field into a dict composed of the original editor name and a editor id (without coma or blank).

Parameters **record** (*dict*) – the record.

Returns dict – the modified record.

customization.**journal**(*record*)

Turn the journal field into a dict composed of the original journal name and a journal id (without coma or blank).

Parameters **record** (*dict*) – the record.

Returns dict – the modified record.

customization.**keyword**(*record*, *sep=', |;'*)

Split keyword field into a list.

Parameters

- **record** (*string, optional*) – the record.
- **sep** – pattern used for the splitting regexp.

Returns dict – the modified record.

customization.**link**(*record*)

Parameters **record** (*dict*) – the record.

Returns dict – the modified record.

customization.**page_double_hyphen**(*record*)

Separate pages by a double hyphen (–).

Parameters **record** (*dict*) – the record.

Returns dict – the modified record.

customization.**doi**(*record*)

Parameters **record** (*dict*) – the record.

Returns dict – the modified record.

customization.**type**(*record*)
Put the type into lower case.

Parameters **record** (*dict*) – the record.

Returns dict – the modified record.

customization.**convert_to_unicode**(*record*)
Convert accent from latex to unicode style.

Parameters **record** (*dict*) – the record.

Returns dict – the modified record.

customization.**homogenize_latex_encoding**(*record*)
Homogenize the latex enconding style for bibtex

This function is experimental.

Parameters **record** (*dict*) – the record.

Returns dict – the modified record.

customization.**add_plaintext_fields**(*record*)
For each field in the record, add a *plain_* field containing the plaintext, stripped from braces and similar. See https://github.com/sciunto-org/python-bibtexparser/issues/116.

Parameters **record** (*dict*) – the record.

Returns dict – the modified record.

3.4.1 Exception classes

class customization.**InvalidName**
Exception raised by *customization.splitname()* when an invalid name is input.

3.5 bibtexparser.bwriter — Tune the default writer

class bwriter.**BibTexWriter**(*write_common_strings=False*)
Writer to convert a BibDatabase object to a string or file formatted as a BibTeX file.

Example:

```
from bibtexparser.bwriter import BibTexWriter

bib_database = ...

writer = BibTexWriter()
writer.contents = ['comments', 'entries']
writer.indent = '  '
writer.order_entries_by = ('ENTRYTYPE', 'author', 'year')
bibtex_str = bibtexparser.dumps(bib_database, writer)
```

align_values = None
Align values. Determines the maximal number of characters used in any fieldname and aligns all values

comma_first = None
BibTeX syntax allows comma first syntax (common in functional languages), use this to enable comma first syntax as the bwritter output

common_strings = None
Whether common strings are written

contents = None
List of BibTeX elements to write, valid values are *entries*, *comments*, *preambles*, *strings*.

display_order = None
Tuple of fields for display order in a single BibTeX entry. Fields not listed here will be displayed alphabetically at the end. Set to '[]' for alphabetical order. Default: '[]'

entry_separator = None
Characters(s) for separating BibTeX entries. Default: new line.

indent = None
Character(s) for indenting BibTeX field-value pairs. Default: single space.

order_entries_by = None
Tuple of fields for ordering BibTeX entries. Set to *None* to disable sorting. Default: BibTeX key *('ID',)*.

write(*bib_database*)
Converts a bibliographic database to a BibTeX-formatted string.

Parameters **bib_database** (BibDatabase) – bibliographic database to be converted to a BibTeX string

Returns BibTeX-formatted string

Return type str or unicode

3.6 bibtexparser.bibtexexpression — Parser's core relying on pyparsing

CHAPTER 4

How to report a bug?

Bugs can be reported on github or via private communications.

4.1 Steps

1. Make a minimal code, which reproduces the problem.
2. Provide the code, the bibtex (if necessary), the output.
3. For a parsing error, provide the expected output.
4. For a crash, set the logger to the debug level (see below).

If you want to provide a patch (that's wonderful! thank you), please, take few minutes to write a unit test that fails without your contribution.

4.2 Logging module to understand failures

Syntax of bibtex files is simple but there are many possible variations. This library probably fails for some of them.

Bibtexparser includes a large quantity of debug messages which helps to understand why and where the parser fails. The example below can be used to print these messages in the console.

```
import logging
import logging.config

logger = logging.getLogger(__name__)

logging.config.dictConfig({
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'standard': {
```

```
            'format': '%(asctime)s [%(levelname)s] %(name)s %(funcName)s:%(lineno)d:
↪%(message)s'
        },
    },
    'handlers': {
        'default': {
            'level':'DEBUG',
            'formatter': 'standard',
            'class':'logging.StreamHandler',
        },
    },
    'loggers': {
        '': {
            'handlers': ['default'],
            'level': 'DEBUG',
            'formatter': 'standard',
            'propagate': True
        }
    }
})


if __name__ == '__main__':
    bibtex = """@ARTICLE{Cesar2013,
      author = {Jean César},
      title = {An amazing title},
      year = {2013},
      month = jan,
      volume = {12},
      pages = {12--23},
      journal = {Nice Journal},
      abstract = {This is an abstract. This line should be long enough to test
             multilines...},
      comments = {A comment},
      keywords = {keyword1, keyword2},
    }
    """

    with open('/tmp/bibtex.bib', 'w') as bibfile:
        bibfile.write(bibtex)

    from bibtexparser.bparser import BibTexParser

    with open('/tmp/bibtex.bib', 'r') as bibfile:
        bp = BibTexParser(bibfile.read())
        print(bp.get_entry_list())
```

I recommend you to use this output if you would like to report a bug.

CHAPTER 5

Bibtex tips, conventions and unrelated projects

This page presents various resources about bibtex in general.

5.1 Format

http://maverick.inria.fr/~Xavier.Decoret/resources/xdkbibtex/bibtex_summary.html

- Comments
- Variable
- @preamble
- Name convention

5.1.1 Upper case letters in titles

Put the letter/word in curly braces like {this}.

5.1.2 General references

- http://tug.ctan.org/tex-archive/info/bibtex/tamethebeast/ttb_en.pdf
- http://ctan.mirrors.hoobly.com/macros/latex/contrib/biblatex/doc/biblatex.pdf

5.1.3 IEEE citation reference

- https://origin.www.ieee.org/documents/ieeecitationref.pdf

5.1.4 Common Errors in Bibliographies John Owens

- http://www.ece.ucdavis.edu/~jowens/biberrors.html

5.1.5 Common abbreviations for journals

- Jabref list http://jabref.sourceforge.net/resources.php#downloadlists

5.2 Projects

Here are some interesting projects using bibtex but not necessarily this parser.

5.2.1 Display your bibliography in html pages

- http://www.monperrus.net/martin/bibtexbrowser/

CHAPTER 6

Who uses BibtexParser?

If your project uses BibtexParser, you can ask for the addition of a link in this list.

- http://timotheepoisot.fr/2013/11/10/shared-bibtex-file-markdown/
- https://github.com/Phyks/BMC
- http://aurelien.naldi.info/research/publications.html
- http://robot.kut.ac.kr/publications
- https://git.atelo.org/etlapale/bibgen
- https://onmenwhostareongraphs.wordpress.com/2015/06/09/graph-display-software-for-author-relationships-with-bibtex-files/
- https://github.com/vitorfs/parsifal

CHAPTER 7

Other projects

- http://pybtex.sourceforge.net/
- http://pybliographer.org/
- https://github.com/matthew-brett/babybib

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

c

Index