# Contents

1 How to install and test? 3
  1.1 How to install? ................................................................. 3
  1.2 How to run the test suite? .............................................. 4

2 Tutorial 5
  2.1 Step 0: Vocabulary .......................................................... 5
  2.2 Step 1: Prepare a BibTeX file ........................................ 5
  2.3 Step 2: Parse it! ............................................................. 6
  2.4 Step 3: Export ............................................................... 7
  2.5 Step 4: Add salt and pepper ......................................... 9

3 bibtexparser: API 13
  3.1 bibtexparser — Parsing and writing BibTeX files ............ 13
  3.2 bibtexparser.bibdatabase — The bibliographic database object ........................................ 14
  3.3 bibtexparser.bparser — Tune the default parser .......... 15
  3.4 bibtexparser.customization — Functions to customize records ........................................ 16
  3.5 bibtexparser.bwriter — Tune the default writer ......... 19
  3.6 bibtexparser.bibtexexpression — Parser’s core relying on pyparsing ................................. 20

4 How to report a bug? 23
  4.1 Steps ................................................................. 23
  4.2 Logging module to understand failures .......................... 23

5 Bibtex tips, conventions and unrelated projects 25
  5.1 Format ................................................................. 25
  5.2 Projects ............................................................... 26

6 Who uses BibtexParser? 27

7 Other projects 29

8 Indices and tables 31

Python Module Index 33
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:
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 bibtexparserr
pip install -r requirements.txt
python setup.py install
nose
test
```

If you already have a virtualenv, you can use workon

```
pew workon bibtexparserr
```

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 :) 
```
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},
  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}
}
```

(continues on next page)
2.3 Step 2: Parse it!

2.3.1 Simplest call

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

```python
import bibexpser

with open('bibtex.bib') as bibtex_file:
    bib_database = bibexpser.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',
  'abstract': 'This is an abstract. This line should be long enough to
  test\multlines...',
  '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)
```

Note: If your bibtex contains months defined as strings such as `month = jan`, you will need to parse it with the `common_strings` option: `bib_database = bibexpser.bparser.BibTexParser(common_strings=True).parse_file(bibtex_file)`. (More in Using bibtex strings.)

2.3.2 Parse a string

If for some reason, you prefer to parse a string, that’s also possible:
2.3.3 Tune parser's options

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

```python
import bibtxexparser
from bibtxexparser.bparser import BibTexParser

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

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

*Note:* The `common_strings` option needs to be set when the parser object is created and has no effect if changed afterwards.

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:

```python
import bibtxexparser

bibtex_str = bibtxexparser.dumps(bib_database)
```

or a BibTeX file like this:

```python
import bibtxexparser

with open('bibtex.bib', 'w') as bibtex_file:
    bibtxexparser.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
multilines...',
     '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
multilines...',
     'title': 'An amazing title',
     'year': '2013',
     'volume': '12',
     'ID': 'Cesar2013',
     'author': 'Jean César',
     'keyword': 'keyword1, keyword2',
     'ENTRYTYPE': 'article'}
]

(continues on next page)
This code results in the following, customized, file:

```latex
@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 *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.

```python
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
```

(continued on next page)
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.

```python
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.

```python
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.

```python
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

- bibtexparser: API
  - bibtexparser — Parsing and writing BibTeX files
  - bibtexparser.bibdatabase — The bibliographic database object

2.5. Step 4: Add salt and pepper
– bibtexparser.bparser — Tune the default parser
– bibtexparser.customization — Functions to customize records
  * Exception classes
– bibtexparser.bwriter — Tune the default writer
– bibtexparser.bibtexexpression — Parser’s core relying on pyparsing
3.1 bibtxparser — Parsing and writing BibTeX files

BibTeX is a bibliographic data file format.

The bibtxparser 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:

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

And writing:

```python
import bibtxparser
with open('bibtex.bib', 'w') as bibtex_file:
    bibtxparser.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 bibtexparser.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 bibtexparser.bparser.BibTexParser(data=None, customization=None, ignore_nonstandard_types=True, homogenize_fields=False, interpolate_strings=True, common_strings=False, add_missing_from_crossref=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 bib-
tex 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** – bool (default False) Include common string definitions (e.g. month
abbreviations) to the bibtex file.

• **add_missing_from_crossref** – bool (default False) Resolve BibTeX references set
in the crossref field for BibTeX entries and add the fields from the referenced entry to the
referencing entry.

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

---

3.3. *bibtexparser.bparser — Tune the default parser*
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. If False, an exception is raised.

Type str or unicode
Type boolean
Returns bibliographic database
Return type 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. If False, an exception is raised.

Type file
Type boolean
Returns bibliographic database
Return type BibDatabase

3.4 bibtexparsercustomization — 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.

bibtexparsercustomization.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

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

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

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

**Returns** list – Correctly formatted names

---

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

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

Split author field into a list of “Name, Surname”.

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

**Returns** dict – the modified record.

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

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

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

**Returns** dict – the modified record.

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

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

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

**Returns** dict – the modified record.

**bibtexparser.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.

*bibtexparser.customization.link(record)*

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

**Returns**  
dict – the modified record.

*bibtexparser.customization.page_double_hyphen(record)*

Separate pages by a double hyphen (–).

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

**Returns**  
dict – the modified record.

*bibtexparser.customization.doi(record)*

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

**Returns**  
dict – the modified record.

*bibtexparser.customization.type(record)*

Put the type into lower case.

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

**Returns**  
dict – the modified record.

*bibtexparser.customization.convert_to_unicode(record)*

Convert accent from latex to unicode style.

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

**Returns**  
dict – the modified record.

*bibtexparser.customization.homogenize_latex_encoding(record)*

Homogenize the latex encoding style for bibtex

This function is experimental.

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

**Returns**  
dict – the modified record.

*bibtexparser.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**  
bibtexparser.customization.InvalidName

Exception raised by customization.splitname() when an invalid name is input.
3.5 `bibtexparser.bwriter` — Tune the default writer

```python
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)
```

**add_trailing_comma** = None

BibTeX syntax allows the comma to be optional at the end of the last field in an entry. Use this to enable writing this last comma in the bwriter output. Defaults: False.

**comma_first** = None

BibTeX syntax allows comma first syntax (common in functional languages), use this to enable comma first syntax as the bwriter 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 \texttt{bibtexpars}.\texttt{bibtexexpression} — Parser’s core relying on \texttt{pyparsing}

\texttt{class \ bibtexpars}.\texttt{bibtexexpression}.\texttt{BibtexExpression}

\texttt{Gives access to pyparsing expressions.}

Attributes are \texttt{pyparsing} expressions for the following elements:

- \texttt{main_expression}: the \texttt{bibtex} file
- \texttt{string_def}: a string definition
- \texttt{preamble_decl}: a preamble declaration
- \texttt{explicit_comment}: an explicit comment
- \texttt{entry}: an entry definition
- \texttt{implicit_comment}: an implicit comment

\texttt{exception \ Exception (pstr, loc=0, msg=None, elem=None)}

Exception thrown when parse expressions don’t match class; supported attributes by name are:

- \texttt{lineno} - returns the line number of the exception text
- \texttt{col} - returns the column number of the exception text
- \texttt{line} - returns the line containing the exception text

\texttt{Example::}

\begin{verbatim}
try: \texttt{Word(nums).setName("integer").parseString("ABC")}
except \texttt{ParseException as pe: \ print(pe) \ print("column: \{\}".format(pe.col))}
\end{verbatim}

\texttt{prints::}

\texttt{Expected integer (at char 0), (line:1, col:1) column: 1}

\texttt{add_log_function (log_fun)}

\texttt{Add notice to logger on entry, comment, preamble, string definitions.}

\texttt{Parameters log_fun – logger function}

\texttt{set_string_expression_parse_action (fun)}

\texttt{Set the parseAction for string_expression expression.}

\texttt{Note: \ See set_string_name_parse_action.}

\texttt{set_string_name_parse_action (fun)}

\texttt{Set the parseAction for string name expression.}

\texttt{Note: \ For some reason \texttt{pyparsing} duplicates the string_name expression so setting its parseAction a posteriori has no effect in the context of a string expression. This is why this function should be used instead.}

\texttt{bibtexpars}.\texttt{bibtexexpression}.\texttt{add_logger_parse_action (expr, log_func)}

\texttt{Register a callback on expression parsing with the adequate message.}
bibtexparser.bibtexexpression.\texttt{field_to_pair}(string, location, token)

Looks for parsed element named ‘Field’.

\textbf{Returns} (name, value).

bibtexparser.bibtexexpression.\texttt{in_braces_or_pars}(exp)

exp \rightarrow (exp)|{exp}

bibtexparser.bibtexexpression.\texttt{strip_after_new_lines}(s)

Removes leading and trailing whitespaces in all but first line.

\textbf{Parameters} s – string or BibDataStringExpression
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. Bibtxparser 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.

```python
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 - %(message)s'
        }
    }
})
```
'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 multiline...},
  comments = {A comment},
  keywords = {keyword1, keyword2},
}
'''

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

  from bibtexparserset importer 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.
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


### 5.1.3 IEEE citation reference

5.1.4 Common Errors in Bibliographies John Owens


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.

- 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://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

bibexparsert, 13
bibexparsert.bibtexexpessrt, 20
bibexparsert.bparser, 15
bibexparsert.customization, 16
Index

A
add_log_function() (bibtexparser.bibtexexpression.BibtexExpression method), 20
add_logger_parse_action() (in module bibtexparser.bibtexexpression), 20
add_plaintext_fields() (in module bibtexparser.customization), 18
add_trailing_comma (bibtexparser.bwriter.BibTexWriter attribute), 19
author() (in module bibtexparser.customization), 17

B
BibDatabase (class in bibtexparser.bibdatabase), 14
BibtexExpression (class in bibtexparser.bibtexexpression), 20
BibtexExpression.ParseException, 20
BibTexParser (class in bibtexparser.bparser), 15
bibtexparser (module), 13
bibtexparser.bibtexexpression (module), 20
bibtexparser.bparser (module), 15
bibtexparser.customization (module), 16
BibTexWriter (class in bibtexparser.bwriter), 19

C
comma_first (bibtexparser.bwriter.BibTexWriter attribute), 19
comments (bibtexparser.bibdatabase.BibDatabase attribute), 14
common_strings (bibtexparser.bparser.BibTexParser attribute), 15
common_strings (bibtexparser.bwriter.BibTexWriter attribute), 19
contents (bibtexparser.bwriter.BibTexWriter attribute), 19
convert_to_unicode() (in module bibtexparser.customization), 18
customization (bibtexparser.bparser.BibTexParser attribute), 15

d
display_order (bibtexparser.bwriter.BibTexWriter attribute), 19
doi() (in module bibtexparser.customization), 18
dump() (in module bibtexparser), 14
dumps() (in module bibtexparser), 14

E
editor() (in module bibtexparser.customization), 17
entries (bibtexparser.bibdatabase.BibDatabase attribute), 14
entries_dict (bibtexparser.bibdatabase.BibDatabase attribute), 15
entry_separator (bibtexparser.bwriter.BibTexWriter attribute), 19

F
field_to_pair() (in module bibtexparser.bibtexexpression), 20

G
getnames() (in module bibtexparser.customization), 17

H
homogenize_fields (bibtexparser.bparser.BibTexParser attribute), 16
homogenize_latex_encoding() (in module bibtexparser.customization), 18

I
ignore_nonstandard_types (bibtexparser.bparser.BibTexParser attribute), 16
in_braces_or_pars() (in module bibtexparser.bibtexexpression), 21
indent (bibtexparser.bwriter.BibTexWriter attribute), 19
interpolate_strings (bibtexparser.bparser.BibTexParser attribute), 16
InvalidName (class in bibtexparser.customization), 18
**J**
njournal() (in module bibexparsr.customization), 17

**K**
keyword() (in module bibexparsr.customization), 17

**L**
link() (in module bibexparsr.customization), 18
load() (in module bibexparsr), 13
loads() (in module bibexparsr), 14

**O**
order_entries_by (bibexparsr.bwriter.BibTexWriter attribute), 19

**P**
page_double_hyphen() (in module bibexparsr.customization), 18
parse() (bibexparsr.bparser.BibTexParser method), 16
parse_file() (bibexparsr.bparser.BibTexParser method), 16
preambles (bibexparsr.bibdatabase.BibDatabase attribute), 15

**S**
set_string_expression_parse_action() (bibexparsr.bibtexexpression.BibtexExpression method), 20
set_string_name_parse_action() (bibexparsr.bibtexexpression.BibtexExpression method), 20
splitname() (in module bibexparsr.customization), 16
strings (bibexparsr.bibdatabase.BibDatabase attribute), 15
strip_after_new_lines() (in module bibexparsr.bibtexexpression), 21

**T**
type() (in module bibexparsr.customization), 18

**W**
write() (bibexparsr.bwriter.BibTexWriter method), 19