BibtexParser Documentation

Release latest

F. Boulogne

Aug 16, 2023

Contents

Author Michael Weiss and other contributors.

Source Code github.com project

Bugs github.com

Generated Aug 16, 2023

Version latest

BibtexParser is a python library to parse bibtex files. It is used by more than 1600 open-source repositories.

Contents:

CHAPTER 1

Installation

1.1 Requirements

Bibtexparsers only requirement is a python interpreter which is not yet EOL (currently >= 3.7).

As of version 2.0.0, bibtexparser is a pure-python project (no direct bindings to C libraries). As such, it should be rather easy to install on any platform.

1.2 Installation of current development version

To install the latest version on the main branch (without manually cloning it), run:

```
pip install --no-cache-dir --force-reinstall git+https://github.com/sciunto-org/
↪python-bibtexparser@main
```

1.3 Installation from PyPI

Warning: Installation of v2 via PyPI is not yet supported. We will start releasing v2 pre-versions soon, then you'll be able to use the installation method below. Until then, please use the "installation of current development version" as described above.

To install the latest release candidate (currently required to use v2) using pip:

```
pip install --pre bibtexparser
```

without the `--pre` option, you will get the latest *v1* version. It has a different API and is not directly compatible with v2.

1.4 Installation from source

Download the source from Github. Navigate to the root of the project and run the following command:

```
pip install .
```

Or, if you want to install dev dependencies:

```
pip install .[test,lint,docs]
```

CHAPTER 2

Quickstart

This section provides a TLDR-style overview of the high-level features of bibtexparser. For more detailed information, please refer to the corresponding sections of the documentation.

2.1 Prerequisite: Vocabulary

- An **entry** refers to a citable item, e.g. *@book{...}*, *@article{...}*, etc.
- A **preamble** is a *@preamble{...}* block.
- A **string** is *@string{...}*.
- An **explicit comment** is written as *@comment{...}*.
- An **implicit comment** is any text not within any *@...{...}* block.
- Each of the above is called a **block**, i.e., any .bib file is a collection of blocks of the above types.

In an entry, you can find

- an **entry type** like *article*, *book*, etc.
- and **entry key**, e.g. *Cesar2013* in *@article{Cesar2013, ...}*.
- and **fields**, which are the key-value pairs in the entry, e.g. *author = {Jean César}*.
- each field has a **field key** and a **field value**.

2.2 Step 1: Parsing with Defaults

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

```
bibtex_str = """
@comment{
    This is my example comment.
}

@ARTICLE{Cesar2013,
  author = {Jean César},
  title = {An amazing title},
  year = {2013},
  volume = {12},
  pages = {12--23},
  journal = {Nice Journal}
}
"""
```

Let's attempt to parse this string using the default bibtexparser configuration:

```
import bibtexparser
library = bibtexparser.parse_string(bibtex_str) # or bibtexparser.parse_file("my_file.
↪bib")
```

The returned *library* object provides access to the parsed blocks, i.e., parsed high-level segments of the bibtex such as entries, comments, strings and preambles. You can access them by type, or iterate over all blocks, as shown below:

```
print(f"Parsed {len(library.blocks)} blocks, including:"
  f"\n\t{len(library.entries)} entries"
    f"\n\t{len(library.comments)} comments"
    f"\n\t{len(library.strings)} strings and"
    f"\n\t{len(library.preambles)} preambles")

# Output:
# Parsed 2 blocks, including:
#   1 entries
#   1 comments
#   0 strings and
#   0 preambles
```

As you can see, the parsed blocks are represented as dedicated object types (entries, strings, preambles and comments). They share some supertype attributes (e.g. they provide access to their raw bibtex representation and their start line in the file), but primarily expose attributes specific to their type (e.g. entries provide access to their key, type and fields).

Example of exposed attributes:

```
# Comments have just one specific attribute
first_comment = library.comments[0]
first_comment.comment # The comment string

# Entries have more attributes
first_entry = library.entries[0]
first_entry.key # The entry key
first_entry.entry_type # The entry type, e.g. "article"
first_entry.fields # The entry fields (e.g. author, title, etc. with their values)
first_entry.fields_dict # The entry fields, as a dictionary by field key

# Each field of the entry is a `bibtexparser.model.Field` instance
first_field = first_entry.fields[0]
first_field.key # The field key, e.g. "author"
first_field.value # The field value, e.g. "Albert Einstein and Boris Johnson"
```

For a list of all available attributes, see the documentation of the *bibtexparser.model* module.

2.3 Step 2: Error Checking

We aim at being as forgiving as possible when parsing BibTeX files: If the parsing of a block fails, we try to recover and continue parsing the rest of the file.

Failed blocks are still stored in the library, and you should check for their presence to make sure mistakes are not going undetected.

```
if len(library.failed_blocks) > 0:
    print("Some blocks failed to parse. Check the entries of `library.failed_blocks`.
↪")
else:
    print("All blocks parsed successfully")
```

Obviously, in your code, you may want to go beyond simply printing a statement when faced with failed_blocks. Here, the actual failed blocks provided in *library.failed_blocks* will provide you some more information (exceeding this tutorial, see the corresponding section of the docs for more detail).

2.4 Step 3: Exporting with Defaults

Eventually, you may want to write the parsed BibTeX back to a file or bibtex string.

This can be quickly achieved using the following:

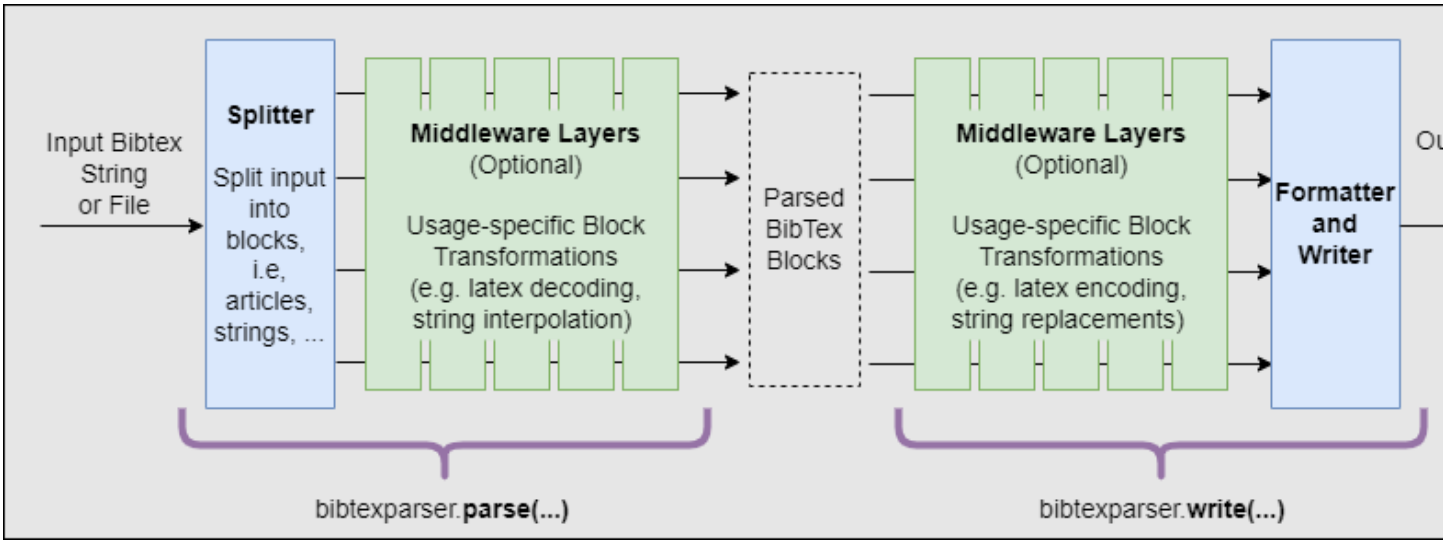
```
new_bibtex_str = bibtexparser.write_string(library) # or bibtexparser.write_file("my_
↪new_file.bib", library)
print(new_bibtex_str)

# Output:
# @comment{This is my example comment.}
#
#
# @article{Cesar2013,
#     author = {Jean César},
#     title = {An amazing title},
#     year = {2013},
#     volume = {12},
#     pages = {12--23},
#     journal = {Nice Journal}
# }
```

As you can see, the content (besides some white-spacing and other layout) is identical to the original string. Naturally, the writer can be configured to your needs. For more information on that, see *the customization documentation*.

CHAPTER 3

Customizing



The core functionality of bibtexparser is deliberately kept simple:

- Upon parsing, the input string is merely split into different parts (blocks) and corresponding subparts (fields, keys, ...).
- Upon writing, the splitting is reversed and the blocks are joined together again, with few formatting options.

Advanced transformations of blocks, such as sorting, encoding, cross-referencing, etc. are not part of the core functionality, but can be optionally added to the parse stack by using the corresponging middleware layers: Middleware layers helper classes providing the functionality take a library object and return a new, transformed version of said library.

3.1 Middleware Layers

```
import bibtexparser.middlewares as m

# We want to add three new middleware layers to our parse stack:
layers = [
    m.MonthIntMiddleware(True), # Months should be represented as int (0-12)
    m.SeparateCoAuthors(True), # Co-authors should be separated as list of strings
    m.SplitNameParts(True) # Individual Names should be split into first, von, last,␣
↪jr parts
]
library = bibtexparser.parse_file('bibtex.bib', append_middleware=layers)
```

This example adds three new middleware layers to the parse stack:

1. The first layer converts the month field (which may be represented as String ("February"), native string reference (feb) or integer (2) to the integer representation (0-12).
2. The second layer splits the author field into a list of authors (and similarly for editors, translators, etc.).
3. The third layer splits the author names into a object representing the first, von, last and jr parts of the name.

3.1.1 Default Parse-Stack

BibtexParser foresees a default parse stack; i.e., some middleware is automatically applied as we assume it to be part of the expected functionality for most users.

Currently, the default parse stack consists of the following layers:

- `bibtexparser.middlewares.ResolveStringReferencesMiddleware`: De-Reference reference to @string definitions.
- `bibtexparser.middlewares.RemoveEnclosingMiddleware`: Removes enclosing (e.g. curly braces or "") from values.

The default write stack consists of the following layers:

- `bibtexparser.middlewares.AddEnclosingMiddleware`: Encloses values in curly braces where needed.

When specifying their own stack, user get to chose if they want to add to or overwrite the default stack by selecting the corresponding argument when calling `bibtexparser.parse` or `bibtexparser.write`:

- `append_middleware`: Add middleware to the default parse stack (similarly `prepend_middleware` for write stack).
- `parse_stack`: Overwrite the default parse stack (similarly `write_stack` for write stack).

> **Warning:** The default parse and write stacks may change on **minor** version updates and between pre-releases. To reduce the risk of unnoticed changes in parsing stack, critical applications may want to hard-code the full parse stack in their code using `parse_stack` and `write_stack` arguments.

3.1.2 Core Middleware

The following middleware layers are part of the core functionality of bibtexparser and maintained as part of the main repository. The functionality is straightforward from the class names, so we will not go into detail here and refer to

the docstrings of the classes instead.

Middleware Layers Regarding Encoding and Enclosing of Values

- `bibtexparser.middlewares.AddEnclosingMiddleware`
- `bibtexparser.middlewares.RemoveEnclosingMiddleware`
- `bibtexparser.middlewares.LatexEncodingMiddleware`
- `bibtexparser.middlewares.LatexDecodingMiddleware`

Middleware Layers Regarding Value References and Representation

- `bibtexparser.middlewares.ResolveStringReferencesMiddleware`
- `bibtexparser.middlewares.MonthIntMiddleware`
- `bibtexparser.middlewares.MonthAbbreviationMiddleware`
- `bibtexparser.middlewares.MonthLongStringMiddleware`

Middleware Layers Regarding Names

- `bibtexparser.middlewares.SeparateCoAuthors`
- `bibtexparser.middlewares.MergeCoAuthors`
- `bibtexparser.middlewares.SplitNameParts` (requires SeperateCoAuthors to be applied first)
- `bibtexparser.middlewares.MergeNameParts`

Sorting Middleware Layers

- `bibtexparser.middlewares.SortBlocksByTypeAndKeyMiddleware`
- `bibtexparser.middlewares.SortFieldsAlphabeticallyMiddleware`
- `bibtexparser.middlewares.SortFieldsCustomMiddleware`

Note: As opposed to bibtexparser v1, the en- and decoding of latex characters is now handled by a third-party library. Previously, this part was responsible for much of the code complexity and bugs in bibtexparser, and leaving this to an established solution is intended to make the use of bibtexparser much more stable, even if it comes at the cost of slightly reduced functionality and performance. See the migration docs, if you are migrating from bibtexparser v1.

3.1.3 Community-Provided Middleware

Aiming to keep the core functionality of bibtexparser simple, we encourage users to provide their own middleware layers and share them with the community. We will be happy to provide a list of community-provided middleware layers here, so please let us know if you have written one!

Note: To write your own middleware, simply extend the `bibtexparser.middlewares.Blockmiddleware` (for functions working on blocks individually, such as encoding) or `bibtexparser.middlewares.LibraryMiddleware` (for library-wide transformations, such as sorting blocks) and implement the superclass methods according to the python docstrings. Make sure to check out some core middleware layers for examples.

3.1.4 Metadata Fields

All blocks have a `metadata` attribute, which is a dictionary of arbitrary middleware-value pairs. This is intended for middleware layers to store metadata about the transformation made by them, which in turn can be used by other middleware layers (e.g. to reverse the transformation).

The metadata attribute and its exact specification is still experimental and subject to breaking changes even within minor/path versions. Even when not experimental anymore, it is not intended to be used by users directly, and may be changed as needed by the corresponding middleware maintainers.

3.2 Formatting Options for Writing

Basic formatting options (e.g. indentation, line breaks, etc.) have no influence on the `bibtexparser.bparser.Library` representation and should not / cannot therefore be specified as middleware layers. These options are instead specified as arguments to the `bibtexparser.write` function. Specifically, a user may pass a `bibtexparser.BibtexFormatter` object to the `bibtex_format` argument of `bibtexparser.write`.

```
bibtex_format = bibtexparser.BibtexFormat()
bibtex_format.indent = '    '
bibtex_format.block_separator = '\n\n'
bib_str = bibtexparser.write_string(library, bibtex_format=bibtex_format)
```

A few more options are provided and we refer to the docstrings of `bibtexparser.BibtexFormat` for details. Note: Sorting of blocks and fields is done with the corresponding middleware, as described above.

CHAPTER 4

Migrating: v1 -> v2

Before you start migrating, we recommend you read the docs regarding the terminology and architecture of bibtex-parser v2, and have a quick look at the tutorial to get a feeling for the new API.

4.1 Status of *v2*

The *v2* branch is well tested and reasonably stable, but it is not yet widely adopted - as an early adopter, y you may encounter some bugs. If you do, please report them on the issue tracker. Also, note that some interfaces may change sightly before we release v2.0.0 as stable.

Some customizations from *v1* are not implemented in *v2*, as we doubt they are widely used. If you need one of these features, please let us know on the issue tracker.

4.2 Differences between v1 and v2

From a user perspective *v2* has the following advantages over *v1*:

- Order of magnitudes faster
- Easily customizable parsing and writing
- Access to more information, such as raw, unparsed bibtex.
- Fault-Tolerant: Able to parse files with syntax errors
- Robuster handling of de- and encoding (special chars, . . .).
- Permissive MIT license

Implementation-wise, the main difference of *v2* is that it does not depend on *pyparsing* anymore. Also, it does not implement any en-/decoding of special characters, but relies on external libraries for this.

To implement these changes, we had to make some breaking changes to the API. Amongst others, be aware that:

- The used vocabulary has slightly changed. [*docs*]

- The primary entrypoints have changed. [*docs*]
- The module *bibtexparser.customizations* been replaced by the module *bibtexparser.middleware* [*docs*]

4.3 Minimal Migration Guide (without customizations)

The following code snippets show how to migrate from *v1* to *v2* for the most common use cases. It aims to provide the quickest way to get *v1* code running with *v2*. As such, it makes reduced use of the new features of *v2* and makes use of backwards compatibility APIs where possible.

Warning: This migration guide is not complete. It covers the parts which are presumably the trickiest ones to migrate. Further migration steps should be needed, but should either be trivial or very specific to your use case (in the latter case you may want to use at the *customization* docs).

4.3.1 Changing the entrypoint with default settings

To make sure that users dont "migrate by accident" to bibtex v2, we changed the entrypoint of the package:

```
# v1
import bibtexparser
with open('bibtex.bib') as bibtex_file:
    bib_database = bibtexparser.load(bibtex_file)

# v2
import bibtexparser
library = bibtexparser.parse_file(bibtex_file)
```

For most usecases, these default settings should be sufficient, even though there are differences in the default configurations between v1 and v2 and thus the outcome you will see. Read the *customization docs* for instruction on how to customize the parsing behavior.

4.3.2 Accessing the library

While in v1 entries were represented as dicts, in v2 they are represented as *Entry* objects.

```
# v1
for entry in bib_database.entries:
    print(entry['title'])

# v2
for entry in library.entries:
    # ... the new 'typed' way to access fields values ...
    print(entry.fields_dict['title'].value)
    # ... but to facilitate migration or simple cases, this shorthand notation also␣
↪works ...
    print(entry['title'])
```

Similarly, other block types (comments, strings, . . .) are now also represented as dedicated *object types*, but for them, the migration is straight forward and we will not go into detail here.

Note: Working with the actual field instances (*entry.fields* or *entry.fields_dict*) and not the shorthand notation (*entry[field_key]*) makes additional information (e.g. raw bibtex or start line of the field in the parsed file) available. We recommend you check out the new data types and their attributes.

4.3.3 Writing a bibtex file (possibly with customizations)

The way to write a bibtex file has changed fundamentally in *v2*, and is now handeled in a fashion very similar to the parsing. See the *writing quickstart* and *writing formatting* for more information.

CHAPTER 5

Biber & Biblatex

Due to its simpel and high-level nature, this library should not only support BibTeX, but also be able to part biblatex and biber files with ease, as they all share the same general syntax.

That said, we did not explicitely check against all of biber and biblatex features. Should you detect anything which is not supported, please open an issue or send a pull request.

Contents

CHAPTER 6

Full API

CHAPTER 7

.bib resources

This page presents various interesting links regarding .bib-based libraries.

7.1 Format & Co.

- Systematic reference manual for the biblatex package
  http://ctan.mirrors.hoobly.com/macros/latex/contrib/biblatex/doc/biblatex.pdf
- IEEE citation reference
  https://origin.www.ieee.org/documents/ieeecitationref.pdf
- “Tame the Beast” by Nicolas Markey
  http://tug.ctan.org/tex-archive/info/bibtex/tamethebeast/ttb_en.pdf
- “Bibtex Summary” by Xavier Décoret
  http://maverick.inria.fr/~Xavier.Decoret/resources/xdkbibtex/bibtex_summary.html
- “Common Errors in Bibliographies” by John Owen
  http://www.ece.ucdavis.edu/~jowens/biberrors.html
- Common abbreviations for journals
  http://jabref.sourceforge.net/resources.php#downloadlists

7.2 Projects

Here are some interesting projects using .bib-based libraries (but not necessarily this parser).

- Clean your bibtex files in the browser, javascript or command-line.
  https://flamingtempura.github.io/bibtex-tidy (browser app)
  https://github.com/FlamingTempura/bibtex-tidy (github repo)
- Display your bibliography in html pages:

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

CHAPTER 8

Indices and tables

- genindex
- modindex
- search