# HEPData Converter Documentation

***Release 0.3.0***

**CERN**

**Mar 12, 2024**

# CONTENTS

This part of the documentation will show you how to get started in using the HEPData Converter.

CHAPTER

ONE

# INSTALLATION

Normal HEPData users and data submitters should not need to install this package, since it is automatically invoked when uploading a single text file with extension `.oldhepdata` to the hepdata.net submission system, or when requesting one of the alternative output formats via the web interface.

To install this package locally, you first need to install YODA and ROOT (including PyROOT). Check that you can `import yoda` and `import ROOT` from Python. You might want to install into a dedicated virtual environment:

```
$ python3 -m venv hepdata-converter
$ source hepdata-converter/bin/activate
(hepdata-converter)$ pip install hepdata-converter
```

This will install the latest released version from PyPI.

## 1.1 Developers

Developers might want to instead install the project directly from GitHub in editable mode:

```
$ git clone https://github.com/HEPData/hepdata-converter
$ cd hepdata-converter
$ python3 -m venv venv
$ source venv/bin/activate
(venv)$ pip install -e '.[tests]'
```

Developers can then run the tests with the following command:

```
python -m unittest discover hepdata_converter/testsuite 'test_*'
```

The documentation can be built locally in the virtual environment using Sphinx:

```
(venv)$ pip install -e '.[docs]'
(venv)$ cd docs
(venv)$ make html
```

Then view the output by opening a web browser at `_build/html/index.html`. Developers should also check that they can successfully build other formats using `make latexpdf` and `make epub`. All three formats will be built by *Read the Docs* for the `main` branch on GitHub.

## 1.2 Docker

Alternatively, a Docker image is available (see the hepdata-converter-docker repository) containing the dependencies such as YODA and ROOT, but not the `hepdata-converter` package itself.

```
$ docker pull hepdata/hepdata-converter
$ docker run --rm -it hepdata/hepdata-converter /bin/bash
```

The `hepdata-converter` package can be installed inside the Docker container:

```
root@617be04cbab5:/# pip install hepdata-converter
root@617be04cbab5:/# hepdata-converter -h
root@617be04cbab5:/# python -c 'import hepdata_converter'
```

Note that the Docker container will be automatically removed when it exits (if running with the `--rm` option). The Python module or CLI can then be used as described in *Usage*. Input and output files can be moved between the local filesystem and the running Docker container using the `docker cp` command, for example,

```
$ docker cp sample.oldhepdata 617be04cbab5:/
root@617be04cbab5:/# hepdata-converter -i oldhepdata sample.oldhepdata SampleYAML
$ docker cp 617be04cbab5:/SampleYAML .
```

where the prompt `$` indicates a terminal corresponding to the local filesystem and the prompt `root@617be04cbab5:/#` indicates another terminal corresponding to the running Docker container.

Alternatively, developers can install from GitHub and mount the current directory of the local filesystem when running the Docker container:

```
$ git clone https://github.com/HEPData/hepdata-converter
$ cd hepdata-converter
$ docker run -v $PWD:$PWD -w $PWD --rm -it hepdata/hepdata-converter /bin/bash
root@2c22e88402d2:/hepdata-converter# pip install -e '.[tests]'
root@2c22e88402d2:/hepdata-converter# hepdata-converter -h
root@2c22e88402d2:/hepdata-converter# python -c 'import hepdata_converter'
root@2c22e88402d2:/hepdata-converter# python -m unittest discover hepdata_converter/
↪testsuite 'test_*'
```

CHAPTER

TWO

# USAGE

The library exposes a single function `convert` which enables conversion from different input formats (`oldhepdata`, `yaml`) to different output formats (`csv`, `root`, `yaml`, `yoda`, `yoda1`), by using a simple in-memory intermediary format.

## 2.1 Python

```
import hepdata_converter

hepdata_converter.convert('sample.oldhepdata', 'Sample', options={'input_format':
↪'oldhepdata', 'output_format': 'yaml'})
```

## 2.2 CLI

```
hepdata-converter -i oldhepdata -o yaml sample.oldhepdata Sample
```

The default input and output formats are `yaml` if not specified explicitly.

See a help message with more detailed options using `hepdata-converter -h`.

CHAPTER

THREE

# API REFERENCE

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

## 3.1 API

hepdata_converter.**convert**(*input*, *output=None*, *options={}*)

Converts a supported `input_format` (*oldhepdata*, *yaml*) to a supported `output_format` (*csv*, *root*, *yaml*, *yoda*, *yoda1*).

**Parameters**

- **input** (*str*) – location of input file for *oldhepdata* format or input directory for *yaml* format
- **output** (*str*) – location of output directory to which converted files will be written
- **options** (*dict*) – additional options such as `input_format` and `output_format` used for conversion

**Raises**

`ValueError` – raised if no `input_format` or `output_format` is specified

## 3.2 Extending the library

To extend the library with new formats (either input or output) a developer only needs to subclass the specified class (for reading, `hepdata_converter.parsers.Parser`, for writing, `hepdata_converter.writers.Writer`), and make sure that files containing these implementations are respectively in `hepdata_converter.parsers` or `hepdata_converter.writers`.

### 3.2.1 Creating a new Parser

In order to create a new Parser you need to create a class inheriting the `Parser` class and override the `def parse(self, data_in, *args, **kwargs):` abstract method. If you're trying to extend the library you should put the file containing the new Parser in the `hepdata_converter/parsers` directory. The name of the class is important: the new parser will be available by this (case-insensitive) name. If your goal is a simple hack, then the package containing the new parser class can be wherever, but the parser class has to be imported before using the `hepdata_converter.convert` function.

An example is given below:

```
from hepdata_converter.common import Option
from hepdata_converter.parsers import Parser, ParsedData


class FOO(Parser):
    help = 'FOO Parser help text displayed in CLI after typing hepdata-converter --help'

    @classmethod
    def options(cls):
        options = Parser.options()
        # add foo_option which is bool and has default value of True
        # it will be automatically added as named argument to __init__ function
        # as foo_option (code below will work):
        # foo = FOO(foo_option=False)
        #
        # additionally it will be accessible inside the class instance as
        # self.foo_option

        options['foo_option'] = Option('foo-option', default=True, type=bool,␣
↪required=False,
                                       help='Description of the option printed in CLI')

    def parse(self, data_in, *args, **kwargs):
        # WARNING it is developers responsibility to be able to handle
        # data_in regardless whether it is string (path) or filelike
        # object

        # list of hepdata_converter.Table objects
        tables = []
        # dictionary corresponding to submission.yaml general element (comment, license -
↪ not table data)
        metadata = {}

        # ... parse data_in into metadata and tables

        return ParsedData(metadata, tables)
```

If this class is put in (e.g.) `hepdata_converter/parsers/foo_parser.py` then it could be accessed from Python code as:

```
import hepdata_converter

hepdata_converter.convert('/path/to/input', '/path/to/output',
                          options={'input_format': 'foo'})
```

It could also be accessed from the CLI:

```
$ hepdata-converter --input-format foo /path/to/input /path/to/output
```

**WARNING**: it is the developer's responsibility to be able to handle `data_in` in `def parse(self, data_in, *args, **kwargs):` regardless whether it is a string (path) or a file-like object.

### 3.2.2 Creating a new Writer

Creation of a new Writer is similar to creating a new Parser (see above), but for the sake of completeness the full description is provided below. In order to create a new Writer you need to create a class inheriting the `Writer` class and override the `def write(self, data_in, data_out, *args, **kwargs):` abstract method. If you're trying to extend the library you should put the file containing the new Parser in the `hepdata_converter/writers` directory. The name of the class is important: the new writer will be available by this (case-insensitive) name. If your goal is a simple hack, then the package containing the new writer class can be wherever, but the writer class has to be imported before using the `hepdata_converter.convert` function.

An example is given below:

```
from hepdata_converter.common import Option
from hepdata_converter.writers import Writer


class FOO(Writer):
    help = 'FOO Writer help text displayed in CLI after typing hepdata-converter --help'

    @classmethod
    def options(cls):
        options = Writer.options()
        # add foo_option which is bool and has default value of True
        # it will be automatically added as named argument to __init__ function
        # as foo_option (code below will work):
        # foo = FOO(foo_option=False)
        #
        # additionally it will be accessible inside the class instance as
        # self.foo_option

        options['foo_option'] = Option('foo-option', default=True, type=bool,␣
↪required=False,
                                       help='Description of the option printed in CLI')

    def write(self, data_in, data_out, *args, **kwargs):
        # data_in is directly passed from Parser.parse method
        # and is instance of ParsedData

        # WARNING it is developers responsibility to be able to handle
        # data_out regardless whether it is string (path) or filelike
        # object

        pass
```

If this class is put in (e.g.) `hepdata_converter/writers/foo_writer.py` then it could be accessed from Python code as:

```
import hepdata_converter

hepdata_converter.convert('/path/to/input', '/path/to/output',
                          options={'output_format': 'foo'})
```

It could also be accessed from the CLI:

```
hepdata-converter --output-format foo /path/to/input /path/to/output
```

**WARNING**: it is the developer's responsibility to be able to handle `data_out` in `def write(self, data_in, data_out, *args, **kwargs):` regardless whether it is a string (path) or a file-like object.

CHAPTER

FOUR

# ADDITIONAL NOTES

This package was started in 2015 by Michał Szostak (@michal-szostak) as a CERN summer student project.

## 4.1 License

# INDEX

## C