
pyproject-parser

Release 0.11.1

Parser for 'pyproject.toml'

Dominic Davis-Foster

May 20, 2024

Contents

| | | |
|----------|--|-----------|
| 1 | Installation | 1 |
| 1.1 | from PyPI | 1 |
| 1.2 | from Anaconda | 1 |
| 1.3 | from GitHub | 1 |
| 2 | CLI | 3 |
| 2.1 | Commands | 3 |
| 2.2 | As a pre-commit hook | 5 |
| 3 | pyproject_parser | 7 |
| 3.1 | PyProject | 7 |
| 3.2 | PyProjectTomlEncoder | 10 |
| 3.3 | _PP | 11 |
| 4 | pyproject_parser.classes | 13 |
| 4.1 | License | 13 |
| 4.2 | Readme | 15 |
| 4.3 | _L | 16 |
| 4.4 | _R | 16 |
| 5 | pyproject_parser.cli | 17 |
| 5.1 | ConfigTracebackHandler | 17 |
| 5.2 | resolve_class | 18 |
| 5.3 | prettify_deprecation_warning | 18 |
| 6 | pyproject_parser.parsers | 19 |
| 6.1 | RequiredKeysConfigParser | 19 |
| 6.2 | BuildSystemParser | 20 |
| 6.3 | PEP621Parser | 21 |
| 7 | pyproject_parser.type_hints | 31 |
| 7.1 | Author | 31 |
| 7.2 | BuildSystemDict | 31 |
| 7.3 | ContentTypes | 31 |
| 7.4 | Dynamic | 32 |
| 7.5 | ProjectDict | 32 |
| 7.6 | ReadmeDict | 32 |
| 8 | pyproject_parser.utils | 33 |
| 8.1 | PyProjectDeprecationWarning | 33 |
| 8.2 | content_type_from_filename | 33 |
| 8.3 | render_markdown | 33 |
| 8.4 | render_rst | 34 |

| | |
|----------------------------|-----------|
| Python Module Index | 35 |
| Index | 37 |

Installation

1.1 from PyPI

```
$ python3 -m pip install pyproject-parser --user
```

1.2 from Anaconda

First add the required channels

```
$ conda config --add channels https://conda.anaconda.org/conda-forge
```

Then install

```
$ conda install pyproject-parser
```

1.3 from GitHub

```
$ python3 -m pip install git+https://github.com/repo-helper/pyproject-parser@master --user
```

`pyproject-parser` also has an optional README validation feature, which checks the README will render correctly on PyPI. This requires that the `readme` extra is installed:

```
$ python -m pip install pyproject-parser[readme]
```

Once the dependencies are installed the validation can be disabled by setting the `CHECK_README` environment variable to 0.

CLI

In addition to the parsing library, `pyproject-parser` has a command-line interface for validating and reformatting `pyproject.toml` files.

New in version 0.2.0.

Attention: This CLI has the following additional requirements:

```
click>=7.1.2
consolekit>=1.4.1
sdjson>=0.3.1
```

These can be installed as follows:

```
$ python -m pip install pyproject-parser[cli]
```

2.1 Commands

2.1.1 check

Validate the given `pyproject.toml` file.

```
pyproject-parser check [OPTIONS] [PYPROJECT_FILE]
```

Options

-P, --parser-class <parser_class>
The class to parse the 'pyproject.toml' file with.

Default `pyproject_parser:PyProject`

-T, --traceback
Show the complete traceback on error.

Arguments

PYPROJECT_FILE

The `pyproject.toml` file.

Optional argument. Default 'pyproject.toml'

The `-P / --parser-class` and `-E / --encoder-class` options must be in the form `<module_name>:<object_name>`. or example, `pyproject_parser:PyProject`, which corresponds to `pyproject_parser.PyProject`. The `module_name` may be any valid Python module, including those containing `.` .

2.1.2 reformat

Reformat the given `pyproject.toml` file.

```
pyproject-parser reformat [OPTIONS] [PYPROJECT_FILE]
```

Options

- E, --encoder-class** <encoder_class>
The class to encode the config to TOML with.

Default `pyproject_parser:PyProjectTomlEncoder`
- d, --show-diff**
Show a (coloured) diff of changes.
- colour, --no-colour**
Whether to use coloured output.
- P, --parser-class** <parser_class>
The class to parse the 'pyproject.toml' file with.

Default `pyproject_parser:PyProject`
- T, --traceback**
Show the complete traceback on error.

Arguments

- PYPROJECT_FILE**
The `pyproject.toml` file.
Optional argument. Default '`pyproject.toml`'

2.1.3 info

New in version 0.5.0.

Extract information from the given `pyproject.toml` file and print the JSON representation.

```
pyproject-parser info [OPTIONS] [FIELD]
```

Options

- r, --resolve**
Resolve file key in `project.readme` and `project.license` (if present) to retrieve the content of the file.

Default `False`
- i, --indent** <indent>
Add indentation to the JSON output.
- f, --file** <pyproject_file>
The `pyproject.toml` file.

-P, --parser-class <parser_class>
The class to parse the 'pyproject.toml' file with.

Default pyproject_parser:PyProject

-T, --traceback
Show the complete traceback on error.

Arguments

FIELD

The field to retrieve from the `pyproject.toml` file.

Optional argument. Default None

Example Usage:

```
# Print the readme text
echo -e $(python3 -m pyproject_parser info project.readme.text -r | tr -d ' ')

# Print the license filename
python3 -m pyproject_parser info project.license.file

# Get one of the project's URLs
python3 -m pyproject_parser info project.urls."Source Code"

# Install the build-system requirements with pip
pip install $(python3 -m pyproject_parser info build-system.requires | jq -r 'join("
↵") ')

# Dump one of the tool sub-tables
python3 -m pyproject_parser info tool.dependency-dash
```

2.2 As a pre-commit hook

pyproject-parser can also be used as a [pre-commit](#) hook. To do so, add the following to your `.pre-commit-config.yaml` file:

```
- repo: https://github.com/repo-helper/pyproject-parser
  rev: 0.11.1
  hooks:
    - id: check-pyproject
    - id: reformat-pyproject
```


pyproject_parser

Parser for `pyproject.toml`.

Classes:

| | |
|---|--|
| <code>PyProject([build_system, project, tool])</code> | Represents a <code>pyproject.toml</code> file. |
| <code>PyProjectTomlEncoder([preserve])</code> | Custom TOML encoder supporting types in <code>pyproject_parser.classes</code> and <code>packaging</code> . |

Data:

| | |
|-------------------|---|
| <code>__PP</code> | Invariant <code>TypeVar</code> bound to <code>pyproject_parser.PyProject</code> . |
|-------------------|---|

class `PyProject` (*build_system=None, project=None, tool={}*)

Bases: `object`

Represents a `pyproject.toml` file.

Parameters

- **build_system** (`Optional[BuildSystemDict]`) – Represents the `build-system` table defined in **PEP 517** and **PEP 518**. Default `None`.
- **project** (`Optional[ProjectDict]`) – Represents the `project` table defined in **PEP 621**. Default `None`.
- **tool** (`Dict[str, Dict[str, Any]]`) – Represents the `tool` table defined in **PEP 518**. Default `{}`.

Methods:

| | |
|---|---|
| <code>__eq__(other)</code> | Return <code>self == other</code> . |
| <code>__repr__()</code> | Return a string representation of the <code>PyProject</code> . |
| <code>dump(filename[, encoder])</code> | Write as TOML to the given file. |
| <code>dumps([encoder])</code> | Serialise to TOML. |
| <code>from_dict(d)</code> | Construct an instance of <code>PyProject</code> from a dictionary. |
| <code>load(filename[, set_defaults])</code> | Load the <code>pyproject.toml</code> configuration mapping from the given file. |
| <code>reformat(filename[, encoder])</code> | Reformat the given <code>pyproject.toml</code> file. |
| <code>resolve_files()</code> | Resolve the <code>file</code> key in <code>readme</code> and <code>license</code> (if present) to retrieve the content of the file. |
| <code>to_dict()</code> | Returns a dictionary containing the contents of the class. |

Attributes:

| | |
|--|---|
| <code>build_system</code> | Represents the <code>build-system</code> table defined in PEP 517 and PEP 518 . |
| <code>build_system_table_parser</code> | The <code>AbstractConfigParser</code> to parse the <code>build-system</code> table with. |
| <code>project</code> | Represents the <code>project</code> table defined in PEP 621 . |
| <code>project_table_parser</code> | The <code>AbstractConfigParser</code> to parse the <code>project</code> table with. |
| <code>tool</code> | Represents the <code>tool</code> table defined in PEP 518 . |
| <code>tool_parsers</code> | A mapping of subtable names to <code>AbstractConfigParser</code> objects to parse the <code>tool</code> table with. |

`__eq__(other)`
Return `self == other`.

Return type `bool`

`__repr__()`
Return a string representation of the `PyProject`.

Return type `str`

build_system

Type: `Optional[BuildSystemDict]`

Represents the `build-system` table defined in [PEP 517](#) and [PEP 518](#).

`build_system_table_parser = <pyproject_parser.parsers.BuildSystemParser object>`

Type: `ClassVar[BuildSystemParser]`

The `AbstractConfigParser` to parse the `build-system` table with.

`dump(filename, encoder=<class 'PyProjectTomlEncoder'>)`

Write as TOML to the given file.

Parameters

- **filename** (`Union[str, Path, PathLike]`) – The filename to write to.
- **encoder** (`Union[Type[TomlEncoder], TomlEncoder]`) – The `TomlEncoder` to use for constructing the output string. Default `pyproject_parser.PyProjectTomlEncoder`.

Return type `str`

Returns A string containing the TOML representation.

`dumps(encoder=<class 'PyProjectTomlEncoder'>)`

Serialise to TOML.

Parameters **encoder** (`Union[Type[TomlEncoder], TomlEncoder]`) – The `TomlEncoder` to use for constructing the output string. Default `pyproject_parser.PyProjectTomlEncoder`.

Return type `str`

classmethod `from_dict(d)`

Construct an instance of *PyProject* from a dictionary.

Parameters `d` (`Mapping[str, Any]`) – The dictionary.

Return type `~_PP`

classmethod `load(filename, set_defaults=False)`

Load the `pyproject.toml` configuration mapping from the given file.

Parameters

- **filename** (`Union[str, Path, PathLike]`)
- **set_defaults** (`bool`) – If `True`, passes `set_defaults=True` the `parse()` method on `build_system_table_parser` and `project_table_parser`. Default `False`.

Return type `~_PP`

project

Type: `Optional[ProjectDict]`

Represents the `project` table defined in **PEP 621**.

project_table_parser = `<pyproject_parser.parsers.PEP621Parser object>`

Type: `ClassVar[PEP621Parser]`

The `AbstractConfigParser` to parse the `project` table with.

classmethod `reformat(filename, encoder=<class 'PyProjectTomlEncoder'>)`

Reformat the given `pyproject.toml` file.

Parameters

- **filename** (`Union[str, Path, PathLike]`) – The file to reformat.
- **encoder** (`Union[Type[TomlEncoder], TomlEncoder]`) – The `TomlEncoder` to use for constructing the output string. Default `pyproject_parser.PyProjectTomlEncoder`.

Return type `str`

Returns A string containing the reformatted TOML.

Changed in version 0.2.0:

- Added the `encoder` argument.
- The parser configured as `project_table_parser` is now used to parse the `project` table, rather than always using `PEP621Parser`.

resolve_files()

Resolve the `file` key in `readme` and `license` (if present) to retrieve the content of the file.

Calling this method may mean it is no longer possible to recreate the original TOML file from this object.

to_dict()

Returns a dictionary containing the contents of the class.

See also: `attr.asdict()`**Return type** `MutableMapping[str, Any]`**tool****Type:** `Dict[str, Dict[str, Any]]`Represents the **tool** table defined in **PEP 518**.**tool_parsers = {}****Type:** `ClassVar[Mapping[str, AbstractConfigParser]]`A mapping of subtable names to `AbstractConfigParser` objects to parse the **tool** table with.For example, to parse `[tool.whey]`:

```
class WheyParser(AbstractConfigParser):
    pass

class CustomPyProject(PyProject):
    tool_parsers = {"whey": WheyParser() }
```

class PyProjectTomlEncoder (*preserve=False*)Bases: `TomlEncoder`Custom TOML encoder supporting types in `pyproject_parser.classes` and `packaging`.**Methods:**

| | |
|---|--|
| <code>dump_packaging_types(obj)</code> | Convert types in <code>packaging</code> to TOML. |
| <code>dumps(table, *, name[, inside_aot])</code> | Serialise the given table. |
| <code>format_inline_array(obj, nest_level)</code> | Format an inline array. |
| <code>format_literal(obj, *[, nest_level])</code> | Format a literal value. |

static dump_packaging_types (*obj*)Convert types in `packaging` to TOML.**Parameters** `obj` (`Union[Version, Requirement, Marker, SpecifierSet]`)**Return type** `str`**dumps** (*table, *, name, inside_aot=False*)

Serialise the given table.

Parameters

- **name** (`str`) – The table name.
- **inside_aot** (`bool`) – Default `False`.

Return type `Iterator[str]`

New in version 0.11.0.

format_inline_array (*obj*, *nest_level*)

Format an inline array.

Parameters

- **obj** (`Union[Tuple, List]`)
- **nest_level** (`int`)

Return type `str`

New in version 0.11.0.

format_literal (*obj*, *, *nest_level=0*)

Format a literal value.

Parameters

- **obj** (`object`)
- **nest_level** (`int`) – Default 0.

Return type `str`

New in version 0.11.0.

_PP = TypeVar(_PP, bound=PyProject)

Type: `TypeVar`

Invariant `TypeVar` bound to `pyproject_parser.PyProject`.

pyproject_parser.classes

Classes to represent readme and license files.

Classes:

| | |
|--|---|
| <code>License([file, text])</code> | Represents a license in PEP 621 configuration. |
| <code>Readme([content_type, charset, file, text])</code> | Represents a readme in PEP 621 configuration. |

Data:

| | |
|-----------------|---|
| <code>_L</code> | Invariant <code>TypeVar</code> bound to <code>pyproject_parser.classes.License</code> . |
| <code>_R</code> | Invariant <code>TypeVar</code> bound to <code>pyproject_parser.classes.Readme</code> . |

class License (*file=None, text=None*)

Bases: `object`

Represents a license in **PEP 621** configuration.

Parameters

- **file** (`Union[str, Path, PathLike, None]`) – The path to the license file. Default `None`.
- **text** (`Optional[str]`) – The content of the license. Default `None`.

Attributes:

| | |
|-------------------|-------------------------------|
| <code>file</code> | The path to the license file. |
| <code>text</code> | The content of the license. |

Methods:

| | |
|---------------------------------|--|
| <code>from_dict(data)</code> | Construct a <code>License</code> from a dictionary containing the same keys as the class constructor. |
| <code>resolve([inplace])</code> | Retrieve the contents of the license file if the <code>file</code> is set. |
| <code>to_dict()</code> | Construct a dictionary containing the keys of the <code>License</code> object. |
| <code>to_pep621_dict()</code> | Construct a dictionary containing the keys of the <code>License</code> object, suitable for use in PEP 621 <code>pyproject.toml</code> configuration. |

file**Type:** `Optional[Path]`

The path to the license file.

classmethod from_dict (*data*)Construct a *License* from a dictionary containing the same keys as the class constructor.Functionally identical to `License(**data)` but provided to give an identical API to *Readme*.**Parameters** *data* (`Mapping[str, str]`)**Return type** `~_L`**See also:** `to_dict()` and `to_pep621_dict()`**resolve** (*inplace=False*)Retrieve the contents of the license file if the *file* is set.Returns a new *License* object with *text* set to the content of the file.**Parameters** *inplace* (`bool`) – Modifies and returns the current object rather than creating a new one.
Default `False`.**Return type** `~_L`**text****Type:** `Optional[str]`

The content of the license.

to_dict ()Construct a dictionary containing the keys of the *License* object.**See also:** `to_pep621_dict()` and `from_dict()`**Return type** `Dict[str, str]`**to_pep621_dict** ()Construct a dictionary containing the keys of the *License* object, suitable for use in **PEP 621** `pyproject.toml` configuration.Unlike `to_dict()` this ignores the `text` key if *self.file* is set.**Return type** `Dict[str, str]`**See also:** `from_dict()`

class Readme (*content_type=None, charset='UTF-8', file=None, text=None*)

Bases: `object`

Represents a readme in **PEP 621** configuration.

Parameters

- **content_type** (`Optional[Literal['text/markdown', 'text/x-rst', 'text/plain']]`) – The content type of the readme. Default `None`.
- **charset** (`str`) – The charset / encoding of the readme. Default `'UTF-8'`.
- **file** (`Union[str, Path, PathLike, None]`) – The path to the license file. Default `None`.
- **text** (`Optional[str]`) – The content of the license. Default `None`.

Attributes:

| | |
|---------------------------|---------------------------------------|
| <code>charset</code> | The charset / encoding of the readme. |
| <code>content_type</code> | The content type of the readme. |
| <code>file</code> | The path to the readme file. |
| <code>text</code> | The content of the readme. |

Methods:

| | |
|---|---|
| <code>from_dict(data)</code> | Construct a <i>Readme</i> from a dictionary containing the same keys as the class constructor. |
| <code>from_file(file[, charset])</code> | Create a <i>Readme</i> from a filename. |
| <code>resolve([inplace])</code> | Retrieve the contents of the readme file if the <code>self.file</code> is set. |
| <code>to_dict()</code> | Construct a dictionary containing the keys of the <i>Readme</i> object. |
| <code>to_pep621_dict()</code> | Construct a dictionary containing the keys of the <i>Readme</i> object, suitable for use in PEP 621 <code>pyproject.toml</code> configuration. |

charset

Type: `str`

The charset / encoding of the readme.

content_type

Type: `Optional[Literal['text/markdown', 'text/x-rst', 'text/plain']]`

The content type of the readme.

file

Type: `Optional[Path]`

The path to the readme file.

classmethod from_dict (*data*)

Construct a *Readme* from a dictionary containing the same keys as the class constructor.

In addition, `content_type` may instead be given as `content-type`.

Parameters *data* (*ReadmeDict*)

Return type `~_R`

See also: `to_dict()` and `to_pep621_dict()`

classmethod `from_file(file, charset='UTF-8')`

Create a *Readme* from a filename.

Parameters

- **file** (`Union[str, Path, PathLike]`) – The path to the readme file.
- **charset** (`str`) – The charset / encoding of the readme file. Default 'UTF-8'.

Return type `~_R`

resolve (`inplace=False`)

Retrieve the contents of the readme file if the `self.file` is set.

Returns a new *Readme* object with `text` set to the content of the file.

Parameters **inplace** (`bool`) – Modifies and returns the current object rather than creating a new one.
Default `False`.

Return type `~_R`

text

Type: `Optional[str]`

The content of the readme.

to_dict ()

Construct a dictionary containing the keys of the *Readme* object.

See also: `to_pep621_dict()` and `from_dict()`

Return type `ReadmeDict`

to_pep621_dict ()

Construct a dictionary containing the keys of the *Readme* object, suitable for use in **PEP 621** `pyproject.toml` configuration.

Unlike `to_dict()` this ignores the `text` key if `self.file` is set, and ignores `self.content_type` if it matches the content-type inferred from the file extension.

See also: `from_dict()`

Return type `Dict[str, str]`

_L = TypeVar(_L, bound=License)

Type: `TypeVar`

Invariant `TypeVar` bound to `pyproject_parser.classes.License`.

_R = TypeVar(_R, bound=Readme)

Type: `TypeVar`

Invariant `TypeVar` bound to `pyproject_parser.classes.Readme`.

pyproject_parser.cli

Command line interface.

New in version 0.2.0.

Attention: This module has the following additional requirements:

```
click>=7.1.2
consolekit>=1.4.1
sdjson>=0.3.1
```

These can be installed as follows:

```
$ python -m pip install pyproject-parser[cli]
```

Classes:

| | |
|---|--|
| <i>ConfigTracebackHandler</i> ([exception]) | <code>consolekit.tracebacks.TracebackHandler</code> which handles <code>dom_toml.parser.BadConfigError</code> . |
|---|--|

Functions:

| | |
|---|--|
| <i>prettify_deprecation_warning</i> () | Catch <i>PyProjectDeprecationWarnings</i> and format them prettily for the command line. |
| <i>resolve_class</i> (raw_class_string, name) | Resolve the class name for the <i>-P</i> / <i>--parser-class</i> and <i>-E</i> / <i>--encoder-class</i> options. |

class ConfigTracebackHandler

Bases: `TracebackHandler`

`consolekit.tracebacks.TracebackHandler` which handles `dom_toml.parser.BadConfigError`.

has_traceback_option = True

Type: bool

Whether to show the message Use '`--traceback`' to view the full traceback. on error. Enabled by default.

New in version 0.5.0: In previous versions this was effectively `False`.

Changed in version 0.6.0: The message is now indented with four spaces.

resolve_class (*raw_class_string*, *name*)

Resolve the class name for the *-P* / *--parser-class* and *-E* / *--encoder-class* options.

Parameters

- **raw_class_string** (*str*)
- **name** (*str*) – The name of the option, e.g. *encoder-class*. Used for error messages.

Return type *Type*

prettyfy_deprecation_warning ()

Catch *PyProjectDeprecationWarnings* and format them prettily for the command line.

New in version 0.5.0.

pyproject_parser.parsers

TOML configuration parsers.

Classes:

| | |
|---|---|
| <code>RequiredKeysConfigParser()</code> | Abstract base class for TOML configuration parsers which have required keys. |
| <code>BuildSystemParser()</code> | Parser for the <code>build-system table</code> from <code>pyproject.toml</code> . |
| <code>PEP621Parser()</code> | Parser for PEP 621 metadata from <code>pyproject.toml</code> . |

class RequiredKeysConfigParser

Bases: `AbstractConfigParser`

Abstract base class for TOML configuration parsers which have required keys.

Methods:

| | |
|---|---|
| <code>parse(config[, set_defaults])</code> | Parse the TOML configuration. |
| <code>assert_sequence_not_str(obj, path[, what])</code> | Assert that <code>obj</code> is a <code>Sequence</code> and not a <code>str</code> , otherwise raise an error with a helpful message. |

parse (*config*, *set_defaults=False*)

Parse the TOML configuration.

Parameters

- **config** (`Dict[str, Any]`)
- **set_defaults** (`bool`) – If `True`, the values in `self.defaults` and `self.factories` will be set as defaults for the returned mapping. Default `False`.

Return type `Dict[str, Any]`

assert_sequence_not_str (*obj*, *path*, *what='type'*)

Assert that `obj` is a `Sequence` and not a `str`, otherwise raise an error with a helpful message.

Parameters

- **obj** (`Any`) – The object to check the type of.
- **path** (`Iterable[str]`) – The elements of the path to `obj` in the TOML mapping.
- **what** (`str`) – What `obj` is, e.g. `'type'`, `'value type'`. Default `'type'`.

class BuildSystemParserBases: *RequiredKeysConfigParser*Parser for the `build-system` table from `pyproject.toml`.**Methods:**

| | |
|---|--|
| <code>normalize_requirement_name(name)</code> | Function to normalize a requirement name per e.g. |
| <code>parse_requires(config)</code> | Parse the <code>requires</code> key. |
| <code>parse_build_backend(config)</code> | Parse the <code>build_backend</code> key defined by PEP 517 . |
| <code>parse_backend_path(config)</code> | Parse the <code>backend-path</code> key defined by PEP 517 . |
| <code>parse(config[, set_defaults])</code> | Parse the TOML configuration. |

static normalize_requirement_name (*name*)Function to normalize a requirement name per e.g. **PEP 503** (where underscores are replaced by hyphens).

New in version 0.9.0.

Return type `str`**parse_requires** (*config*)Parse the `requires` key.**Parameters** `config` (`Dict[str, Any]`) – The unparsed TOML config for the `build-system` table.**Return type** `List[ComparableRequirement]`**parse_build_backend** (*config*)Parse the `build_backend` key defined by **PEP 517**.**Parameters** `config` (`Dict[str, Any]`) – The unparsed TOML config for the `build-system` table.**Return type** `str`**parse_backend_path** (*config*)Parse the `backend-path` key defined by **PEP 517**.**Parameters** `config` (`Dict[str, Any]`) – The unparsed TOML config for the `build-system` table.**Return type** `List[str]`**parse** (*config*, *set_defaults=False*)

Parse the TOML configuration.

Parameters

- **config** (`Dict[str, Any]`)
- **set_defaults** (`bool`) – If `True`, the values in `self.defaults` and `self.factories` will be set as defaults for the returned mapping. Default `False`.

Return type `BuildSystemDict`

class PEP621ParserBases: *RequiredKeysConfigParser*Parser for **PEP 621** metadata from `pyproject.toml`.**Methods:**

| | |
|--|---|
| <code>parse_name(config)</code> | Parse the <code>name</code> key, giving the name of the project. |
| <code>parse_version(config)</code> | Parse the <code>version</code> key, giving the version of the project as supported by PEP 440 . |
| <code>parse_description(config)</code> | Parse the <code>description</code> key, giving a summary description of the project. |
| <code>parse_readme(config)</code> | Parse the <code>readme</code> key, giving the full description of the project (i.e. |
| <code>parse_requires_python(config)</code> | Parse the <code>requires-python</code> key, giving the Python version requirements of the project. |
| <code>parse_license(config)</code> | Parse the <code>license</code> key. |
| <code>parse_authors(config)</code> | Parse the <code>authors</code> key. |
| <code>parse_maintainers(config)</code> | Parse the <code>maintainers</code> key. |
| <code>parse_keywords(config)</code> | Parse the <code>keywords</code> key, giving the keywords for the project. |
| <code>parse_classifiers(config)</code> | Parse the <code>classifiers</code> key, giving the <code>trove classifiers</code> which apply to the project. |
| <code>parse_urls(config)</code> | Parse the <code>urls</code> table. |
| <code>parse_scripts(config)</code> | Parse the <code>scripts</code> table. |
| <code>parse_gui_scripts(config)</code> | Parse the <code>gui-scripts</code> table. |
| <code>parse_entry_points(config)</code> | Parse the <code>entry-points</code> table. |
| <code>normalize_requirement_name(name)</code> | Function to normalize a requirement name per e.g. |
| <code>parse_dependencies(config)</code> | Parse the <code>dependencies</code> key, giving the dependencies of the project. |
| <code>parse_optional_dependencies(config)</code> | Parse the <code>optional-dependencies</code> table, giving the optional dependencies of the project. |
| <code>parse(config[, set_defaults])</code> | Parse the TOML configuration. |

static parse_name (config)Parse the `name` key, giving the name of the project.

- **Format:** `String`
- **Core Metadata:** `Name`

This key is required, and must be defined statically.

Tools SHOULD normalize this name, as specified by **PEP 503**, as soon as it is read for internal consistency.**Example:**

```
[project]
name = "spam"
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.**Return type** `str`

static parse_version (*config*)

Parse the `version` key, giving the version of the project as supported by [PEP 440](#).

- **Format:** `String`
- **Core Metadata:** `Version`

Users SHOULD prefer to specify normalized versions.

Example:

```
[project]
version = "2020.0.0"
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `Version`

parse_description (*config*)

Parse the `description` key, giving a summary description of the project.

- **Format:** `String`
- **Core Metadata:** `Summary`

Example:

```
[project]
description = "Lovely Spam! Wonderful Spam!"
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `str`

static parse_readme (*config*)

Parse the `readme` key, giving the full description of the project (i.e. the README).

- **Format:** `String` or `table`
- **Core Metadata:** `Description`

This field accepts either a string or a table. If it is a string then it is the relative path to a text file containing the full description. The file's encoding MUST be UTF-8, and have one of the following content types:

- `text/markdown`, with a case-insensitive `.md` suffix.
- `text/x-rst`, with a case-insensitive `.rst` suffix.
- `text/plain`, with a case-insensitive `.txt` suffix.

If a tool recognizes more extensions than this PEP, they MAY infer the content-type for the user without specifying this field as dynamic. For all unrecognized suffixes when a content-type is not provided, tools MUST raise an error.

The `readme` field may instead be a table with the following keys:

- `file` – a string value representing a relative path to a file containing the full description.
- `text` – a string value which is the full description.
- `content-type` – (required) a string specifying the content-type of the full description.
- `charset` – (optional, default UTF-8) the encoding of the `file`. Tools MAY support other encodings if they choose to.

The `file` and `text` keys are mutually exclusive, but one must be provided in the table.

Examples:

```
[project]
readme = "README.rst"

[project.readme]
file = "README.rst"
content-type = "text/x-rst"
encoding = "UTF-8"

[project.readme]
text = "Spam is a brand of canned cooked pork made by Hormel Foods_
↳Corporation."
content-type = "text/x-rst"
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `Readme`

static `parse_requires_python` (`config`)

Parse the `requires-python` key, giving the Python version requirements of the project.

The requirement should be in the form of a **PEP 508** marker.

- **Format:** `String`
- **Core Metadata:** `Requires-Python`

Example:

```
[project]
requires-python = ">=3.6"
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `SpecifierSet`

static parse_license (*config*)

Parse the `license` key.

- **Format:** `Table`
- **Core Metadata:** `License`

The table may have one of two keys:

- `file` – a string value that is a relative file path to the file which contains the license for the project. The file’s encoding **MUST** be UTF-8.
- `text` – string value which is the license of the project.

These keys are mutually exclusive, so a tool **MUST** raise an error if the metadata specifies both keys.

Example:

```
[project.license]
file = "LICENSE.rst"

[project.license]
file = "COPYING"

[project.license]
text = """
This software may only be obtained by sending the author a postcard,
and then the user promises not to redistribute it.
"""
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `License`

parse_authors (*config*)

Parse the `authors` key.

- **Format:** `Array` of `inline tables` with string keys and values
- **Core Metadata:** `Author/Author-email`

The tables list the people or organizations considered to be the “authors” of the project.

Each table has 2 keys: `name` and `email`. Both values must be strings.

- The `name` value **MUST** be a valid email name (i.e. whatever can be put as a name, before an email, in **RFC 822**) and not contain commas.
- The `email` value **MUST** be a valid email address.

Both keys are optional.

Using the data to fill in core metadata is as follows:

1. If only `name` is provided, the value goes in `Author`.
2. If only `email` is provided, the value goes in `Author-email`.
3. If both `email` and `name` are provided, the value goes in `Author-email`. The value should be formatted as `{name} <{email}>` (with appropriate quoting, e.g. using `email.headerregistry.Address`).
4. Multiple values should be separated by commas.

Example:

```
[project]
authors = [
    {email = "hi@pradyunsg.me"},
    {name = "Tzu-Ping Chung"}
]

[[project.authors]]
name = "Tzu-Ping Chung"
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `List[Author]`

parse_maintainers (`config`)

Parse the `maintainers` key.

- **Format:** Array of inline tables with string keys and values
- **Core Metadata:** Maintainer/Maintainer-email

The tables list the people or organizations considered to be the “maintainers” of the project.

Each table has 2 keys: `name` and `email`. Both values must be strings.

- The `name` value MUST be a valid email name (i.e. whatever can be put as a name, before an email, in [RFC 822](#)) and not contain commas.
- The `email` value MUST be a valid email address.

Both keys are optional.

1. If only `name` is provided, the value goes in `Maintainer`.
2. If only `email` is provided, the value goes in `Maintainer-email`.
3. If both `email` and `name` are provided, the value goes in `Maintainer-email`, with the format `{name} <{email}>` (with appropriate quoting, e.g. using `email.headerregistry.Address`).
4. Multiple values should be separated by commas.

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `List[Author]`

parse_keywords (`config`)

Parse the `keywords` key, giving the keywords for the project.

- **Format:** Array of strings
- **Core Metadata:** Keywords

Example:

```
[project]
keywords = ["egg", "bacon", "sausage", "tomatoes", "Lobster Thermidor"]
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `List[str]`

parse_classifiers (*config*)

Parse the `classifiers` key, giving the `trove classifiers` which apply to the project.

- **Format:** Array of strings
- **Core Metadata:** Classifiers

Example:

```
[project]
classifiers = [
    "Development Status :: 4 - Beta",
    "Programming Language :: Python"
]
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `List[str]`

parse_urls (*config*)

Parse the `urls` table.

- **Format:** Table, with keys and values of strings
- **Core Metadata:** Project-URL

A table of URLs where the key is the URL label and the value is the URL itself.

Example:

```
[project.urls]
homepage = "https://example.com"
documentation = "https://readthedocs.org"
repository = "https://github.com"
changelog = "https://github.com/me/spam/blob/master/CHANGELOG.md"
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `Dict[str, str]`

parse_scripts (*config*)

Parse the `scripts` table.

Format: Table, with keys and values of strings

The console scripts provided by the project.

The keys are the names of the scripts and the values are the object references in the form `module.submodule:object`.

Example:

```
[project.scripts]
spam-cli = "spam:main_cli"
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `Dict[str, str]`

parse_gui_scripts (*config*)

Parse the `gui-scripts` table.

Format: table, with keys and values of strings

The graphical application scripts provided by the project.

The keys are the names of the scripts and the values are the object references in the form `module.submodule:object`.

Example:

```
[project.gui-scripts]
spam-gui = "spam.gui:main_gui"
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `Dict[str, str]`

parse_entry_points (*config*)

Parse the `entry-points` table.

Format: Table of tables, with keys and values of strings

Each sub-table's name is an entry point group.

- Users MUST NOT create nested sub-tables but instead keep the entry point groups to only one level deep.
- Users MUST NOT created sub-tables for `console_scripts` or `gui_scripts`. Use `[project.scripts]` and `[project.gui-scripts]` instead.

See the [entry point specification](#) for more details.

Example:

```
[project.entry-points."spam.magical"]
tomatoes = "spam:main_tomatoes"

# pytest plugins refer to a module, so there is no ':obj'
[project.entry-points.pytest11]
nbval = "nbval.plugin"
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `Dict[str, Dict[str, str]]`

static normalize_requirement_name (*name*)

Function to normalize a requirement name per e.g. [PEP 503](#) (where underscores are replaced by hyphens).

New in version 0.9.0.

Return type `str`

parse_dependencies (*config*)

Parse the `dependencies` key, giving the dependencies of the project.

- **Format:** Array of [PEP 508](#) strings
- **Core Metadata:** [Requires-Dist](#)

Each string MUST be formatted as a valid [PEP 508](#) string.

Example:

```
[project]
dependencies = [
    "httpx",
    "gidgethub[httpx]>4.0.0",
    "django>2.1; os_name != 'nt'",
    "django>2.0; os_name == 'nt'"
]
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `List[ComparableRequirement]`

parse_optional_dependencies (*config*)

Parse the `optional-dependencies` table, giving the optional dependencies of the project.

- **Format:** Table with values of arrays of [PEP 508](#) strings
- **Core Metadata:** [Requires-Dist](#) and [Provides-Extra](#)
- The keys specify an extra, and must be valid Python identifiers.
- The values are arrays of strings, which must be valid [PEP 508](#) strings.

Example:

```
[project.optional-dependencies]
test = [
    "pytest < 5.0.0",
    "pytest-cov[all]"
]
```

Parameters `config` (`Dict[str, Any]`) – The unparsed TOML config for the `project` table.

Return type `Dict[str, List[ComparableRequirement]]`

Changed in version 0.5.0: Extra names with hyphens are now considered valid. If two extra names would normalize to the same string per [PEP 685](#) a warning is emitted. In a future version this will become an error.

Attention: A future version of *pyproject-parser* will normalize all extra names and write them to METADATA in this normalized form.

parse (*config*, *set_defaults=False*)

Parse the TOML configuration.

Parameters

- **config** (`Dict[str, Any]`)
- **set_defaults** (`bool`) – If `True`, the values in `self.defaults` and `self.factories` will be set as defaults for the returned mapping. Default `False`.

Return type `ProjectDict`

`pyproject_parser.type_hints`

Type hints for `pyproject_parser`.

Classes:

| | |
|------------------------------|--|
| <code>Author</code> | <code>typing.TypedDict</code> representing the items in the <code>authors/maintainers</code> key of PEP 621 . |
| <code>BuildSystemDict</code> | <code>typing.TypedDict</code> representing the output from the <code>BuildSystemParser</code> class. |
| <code>ProjectDict</code> | <code>typing.TypedDict</code> representing the output from the <code>PEP621Parser</code> class. |
| <code>ReadmeDict</code> | <code>typing.TypedDict</code> representing the return type of <code>to_dict()</code> . |

Data:

| | |
|---------------------------|---|
| <code>ContentTypes</code> | Type hint for the valid content-types in the <code>license</code> table defined in PEP 621 . |
| <code>Dynamic</code> | Type hint for the <code>dynamic</code> field defined in PEP 621 . |

typeddict Author

Bases: `TypedDict`

`typing.TypedDict` representing the items in the `authors/maintainers` key of **PEP 621**.

Optional Keys

- **name** (`Optional[str]`)
- **email** (`Optional[str]`)

typeddict BuildSystemDict

Bases: `TypedDict`

`typing.TypedDict` representing the output from the `BuildSystemParser` class.

Required Keys

- **requires** (`List[ComparableRequirement]`)
- **build-backend** (`Optional[str]`)
- **backend-path** (`Optional[List[str]]`)

ContentTypes

Type hint for the valid content-types in the `license` table defined in **PEP 621**.

Alias of `Literal['text/markdown', 'text/x-rst', 'text/plain']`

Dynamic

Type hint for the `dynamic` field defined in [PEP 621](#).

Alias of `Literal['name', 'version', 'description', 'readme', 'requires-python', 'license', 'authors', 'maintainers', 'keywords', 'classifiers', 'urls', 'scripts', 'gui-scripts', 'entry-points', 'dependencies', 'optional-dependencies']`

typeddict ProjectDict

Bases: `TypedDict`

`typing.TypedDict` representing the output from the `PEP621Parser` class.

Required Keys

- **name** (`str`)
- **version** (`Optional[Version]`)
- **description** (`Optional[str]`)
- **readme** (`Optional[Readme]`)
- **requires-python** (`Optional[Marker]`)
- **license** (`Optional[License]`)
- **authors** (`List[Author]`)
- **maintainers** (`List[Author]`)
- **keywords** (`List[str]`)
- **classifiers** (`List[str]`)
- **urls** (`Dict[str, str]`)
- **scripts** (`Dict[str, str]`)
- **gui-scripts** (`Dict[str, str]`)
- **entry-points** (`Dict[str, Dict[str, str]]`)
- **dependencies** (`List[ComparableRequirement]`)
- **optional-dependencies** (`Dict[str, List[ComparableRequirement]]`)
- **dynamic** (`List[Literal['name', 'version', 'description', 'readme', 'requires-python', 'license', 'authors', 'maintainers', 'keywords', 'classifiers', 'urls', 'scripts', 'gui-scripts', 'entry-points', 'dependencies', 'optional-dependencies']]`)

typeddict ReadmeDict

Bases: `TypedDict`

`typing.TypedDict` representing the return type of `to_dict()`.

Optional Keys

- **text** (`str`)
- **file** (`str`)
- **charset** (`str`)
- **content_type** (`Literal['text/markdown', 'text/x-rst', 'text/plain']`)

pyproject_parser.utils

Utility functions.

Exceptions:

| | |
|--|---|
| <code>PyProjectDeprecationWarning</code> | Warning for the use of deprecated features in <i>pyproject.toml</i> . |
|--|---|

Functions:

| | |
|---|---|
| <code>content_type_from_filename(filename)</code> | Return the inferred content type for the given (readme) filename. |
| <code>render_markdown(content)</code> | Attempt to render the given content as Markdown . |
| <code>render_rst(content[, filename])</code> | Attempt to render the given content as ReStructuredText . |

exception `PyProjectDeprecationWarning`

Bases: `Warning`

Warning for the use of deprecated features in *pyproject.toml*.

This is a user-facing warning which will be shown by default. For developer-facing warnings intended for direct consumers of this library, use a standard `DeprecationWarning`.

New in version 0.5.0.

`content_type_from_filename(filename)`

Return the inferred content type for the given (readme) filename.

Parameters `filename` (`Union[str, Path, PathLike]`)

Return type `Literal['text/markdown', 'text/x-rst', 'text/plain']`

`render_markdown(content)`

Attempt to render the given content as [Markdown](#).

Attention: This function has the following additional requirements:

```
docutils>=0.16
readme-renderer[md]>=27.0
```

These can be installed as follows:

```
$ python -m pip install pyproject-parser[readme]
```

Parameters `content` (`str`)

render_rst (*content*, *filename*='<string>')

Attempt to render the given content as `ReStructuredText`.

Attention: This function has the following additional requirements:

```
docutils>=0.16
readme-renderer[md]>=27.0
```

These can be installed as follows:

```
$ python -m pip install pyproject-parser[readme]
```

Parameters

- **content** (`str`)
- **filename** (`Union[str, Path, PathLike]`) – The original filename. Default '<string>'.

Changed in version 0.8.0: Added the `filename` argument.

Python Module Index

p

- `pyproject_parser`, [7](#)
- `pyproject_parser.classes`, [13](#)
- `pyproject_parser.cli`, [17](#)
- `pyproject_parser.parsers`, [19](#)
- `pyproject_parser.type_hints`, [31](#)
- `pyproject_parser.utils`, [33](#)

Symbols

`_L` (in module `pyproject_parser.classes`), 16
`_PP` (in module `pyproject_parser`), 11
`_R` (in module `pyproject_parser.classes`), 16
`__eq__()` (*PyProject* method), 8
`__repr__()` (*PyProject* method), 8
`-E`
 `pyproject-parser-reformat` command
 line option, 4
`-P`
 `pyproject-parser-check` command
 line option, 3
 `pyproject-parser-info` command line
 option, 4
 `pyproject-parser-reformat` command
 line option, 4
`-T`
 `pyproject-parser-check` command
 line option, 3
 `pyproject-parser-info` command line
 option, 5
 `pyproject-parser-reformat` command
 line option, 4
`--colour`
 `pyproject-parser-reformat` command
 line option, 4
`--encoder-class <encoder_class>`
 `pyproject-parser-reformat` command
 line option, 4
`--file <pyproject_file>`
 `pyproject-parser-info` command line
 option, 4
`--indent <indent>`
 `pyproject-parser-info` command line
 option, 4
`--no-colour`
 `pyproject-parser-reformat` command
 line option, 4
`--parser-class <parser_class>`
 `pyproject-parser-check` command
 line option, 3
 `pyproject-parser-info` command line
 option, 4
 `pyproject-parser-reformat` command

 line option, 4
`--resolve`
 `pyproject-parser-info` command line
 option, 4
`--show-diff`
 `pyproject-parser-reformat` command
 line option, 4
`--traceback`
 `pyproject-parser-check` command
 line option, 3
 `pyproject-parser-info` command line
 option, 5
 `pyproject-parser-reformat` command
 line option, 4
`-d`
 `pyproject-parser-reformat` command
 line option, 4
`-f`
 `pyproject-parser-info` command line
 option, 4
`-i`
 `pyproject-parser-info` command line
 option, 4
`-r`
 `pyproject-parser-info` command line
 option, 4

A

`assert_sequence_not_str()`
 (*RequiredKeysConfigParser* method), 19
`Author` (*typeddict* in `pyproject_parser.type_hints`), 31

B

`build_system` (*PyProject* attribute), 8
`build_system_table_parser` (*PyProject*
 attribute), 8
`BuildSystemDict` (*typeddict* in
 `pyproject_parser.type_hints`), 31
`BuildSystemParser` (class in
 `pyproject_parser.parsers`), 20

C

`charset` (*Readme* attribute), 15

ConfigTracebackHandler (*class in*
 pyproject_parser.cli), 17
content_type (*Readme attribute*), 15
content_type_from_filename() (*in module*
 pyproject_parser.utils), 33
ContentTypes (*in module*
 pyproject_parser.type_hints), 31
Core Metadata Field Author, 24
Core Metadata Field
 Author/Author-email, 24
Core Metadata Field Author-email, 24
Core Metadata Field Classifiers, 26
Core Metadata Field Description, 22
Core Metadata Field Keywords, 25
Core Metadata Field License, 24
Core Metadata Field Maintainer, 25
Core Metadata Field
 Maintainer/Maintainer-email, 25
Core Metadata Field Maintainer-email,
 25
Core Metadata Field Name, 21
Core Metadata Field Project-URL, 26
Core Metadata Field Provides-Extra, 28
Core Metadata Field Requires-Dist, 28
Core Metadata Field Requires-Python, 23
Core Metadata Field Summary, 22
Core Metadata Field Version, 22

D

dump() (*PyProject method*), 8
dump_packaging_types()
 (*PyProjectTomlEncoder static method*), 10
dumps() (*PyProject method*), 8
dumps() (*PyProjectTomlEncoder method*), 10
Dynamic (*in module pyproject_parser.type_hints*), 31

F

FIELD
 pyproject-parser-info command line
 option, 5
file (*License attribute*), 13
file (*Readme attribute*), 15
format_inline_array() (*PyProjectTomlEncoder*
 method), 11
format_literal() (*PyProjectTomlEncoder*
 method), 11
from_dict() (*License class method*), 14
from_dict() (*PyProject class method*), 8
from_dict() (*Readme class method*), 15
from_file() (*Readme class method*), 15

H

has_traceback_option (*ConfigTracebackHandler*
 attribute), 17

L

License (*class in pyproject_parser.classes*), 13
load() (*PyProject class method*), 9

M

module
 pyproject_parser, 7
 pyproject_parser.classes, 13
 pyproject_parser.cli, 17
 pyproject_parser.parsers, 19
 pyproject_parser.type_hints, 31
 pyproject_parser.utils, 33

N

normalize_requirement_name()
 (*BuildSystemParser static method*), 20
normalize_requirement_name()
 (*PEP621Parser static method*), 28

P

parse() (*BuildSystemParser method*), 20
parse() (*PEP621Parser method*), 29
parse() (*RequiredKeysConfigParser method*), 19
parse_authors() (*PEP621Parser method*), 24
parse_backend_path() (*BuildSystemParser*
 method), 20
parse_build_backend() (*BuildSystemParser*
 method), 20
parse_classifiers() (*PEP621Parser method*), 26
parse_dependencies() (*PEP621Parser method*),
 28
parse_description() (*PEP621Parser method*), 22
parse_entry_points() (*PEP621Parser method*),
 27
parse_gui_scripts() (*PEP621Parser method*), 27
parse_keywords() (*PEP621Parser method*), 25
parse_license() (*PEP621Parser static method*), 24
parse_maintainers() (*PEP621Parser method*), 25
parse_name() (*PEP621Parser static method*), 21
parse_optional_dependencies()
 (*PEP621Parser method*), 28
parse_readme() (*PEP621Parser static method*), 22
parse_requires() (*BuildSystemParser method*), 20
parse_requires_python() (*PEP621Parser static*
 method), 23
parse_scripts() (*PEP621Parser method*), 26
parse_urls() (*PEP621Parser method*), 26
parse_version() (*PEP621Parser static method*), 21
PEP621Parser (*class in pyproject_parser.parsers*), 21
prettify_deprecation_warning() (*in module*
 pyproject_parser.cli), 18
project (*PyProject attribute*), 9
project_table_parser (*PyProject attribute*), 9

ProjectDict (*typeddict* in *pyproject_parser.type_hints*), 32

PyProject (*class* in *pyproject_parser*), 7

PYPROJECT_FILE

- pyproject-parser-check command line option, 3
- pyproject-parser-reformat command line option, 4

pyproject_parser

- module, 7

pyproject_parser.classes

- module, 13

pyproject_parser.cli

- module, 17

pyproject_parser.parsers

- module, 19

pyproject_parser.type_hints

- module, 31

pyproject_parser.utils

- module, 33

pyproject-parser-check command line option

- P, 3
- T, 3
- parser-class <parser_class>, 3
- traceback, 3
- PYPROJECT_FILE, 3

pyproject-parser-info command line option

- P, 4
- T, 5
- file <pyproject_file>, 4
- indent <indent>, 4
- parser-class <parser_class>, 4
- resolve, 4
- traceback, 5
- f, 4
- i, 4
- r, 4
- FIELD, 5

pyproject-parser-reformat command line option

- E, 4
- P, 4
- T, 4
- colour, 4
- encoder-class <encoder_class>, 4
- no-colour, 4
- parser-class <parser_class>, 4
- show-diff, 4
- traceback, 4
- d, 4
- PYPROJECT_FILE, 4

PyProjectDeprecationWarning, 33

PyProjectTomlEncoder (*class* in *pyproject_parser*), 10

Python Enhancement Proposals

- PEP 440, 21, 22
- PEP 503, 20, 21, 28
- PEP 508, 23, 28
- PEP 517, 7, 8, 20
- PEP 518, 7, 8, 10
- PEP 518#build-system-table, 7, 8, 19, 20
- PEP 518#tool-table, 7, 8, 10
- PEP 621, 7–9, 13–16, 19, 21, 31, 32
- PEP 621#authors-maintainers, 21, 24, 25, 31
- PEP 621#classifiers, 21, 26
- PEP 621#dependencies-optional-dependencies, 21, 28
- PEP 621#description, 21, 22
- PEP 621#dynamic, 31, 32
- PEP 621#entry-points, 21, 27
- PEP 621#gui-scripts, 21, 27
- PEP 621#keywords, 21, 25
- PEP 621#license, 7, 9, 21, 24, 31
- PEP 621#name, 21
- PEP 621#readme, 7, 9, 21, 22
- PEP 621#requires-python, 21, 23
- PEP 621#scripts, 21, 26
- PEP 621#table-name, 7–9, 21–28
- PEP 621#urls, 21, 26
- PEP 621#version, 21, 22
- PEP 685, 28

R

Readme (*class* in *pyproject_parser.classes*), 15

ReadmeDict (*typeddict* in *pyproject_parser.type_hints*), 32

reformat () (*PyProject* class method), 9

render_markdown () (*in module pyproject_parser.utils*), 33

render_rst () (*in module pyproject_parser.utils*), 33

RequiredKeysConfigParser (*class* in *pyproject_parser.parsers*), 19

resolve () (*License* method), 14

resolve () (*Readme* method), 16

resolve_class () (*in module pyproject_parser.cli*), 17

resolve_files () (*PyProject* method), 9

RFC

- RFC 822, 24, 25

T

text (*License* attribute), 14

text (*Readme* attribute), 16

to_dict () (*License* method), 14

`to_dict()` (*PyProject method*), 9
`to_dict()` (*Readme method*), 16
`to_pep621_dict()` (*License method*), 14
`to_pep621_dict()` (*Readme method*), 16
TOML: Array, 24–26, 28
TOML: inline table, 24, 25
TOML: String, 21–23
TOML: string, 25–27
TOML: Table, 24, 26–28
TOML: table, 22, 27
`tool` (*PyProject attribute*), 10
`tool_parsers` (*PyProject attribute*), 10