

pyproject.toml, packaging, and you

Moshe Zadka – <https://cobordism.com>

Acknowledgement of Country

Belmont (in San Francisco Bay Area Peninsula)
Ancestral homeland of the Ramaytush Ohlone people

Matching Donation: Encircle

LGBTQ+ Charity in Utah

<https://encircletogether.org/>

Up to \$100

What is TOML?

Semantics:

What is TOML?

Semantics: JSON + date + float vs. integer

What is TOML?

Semantics: JSON + date + float vs. integer

Syntax:

What is TOML?

Semantics: JSON + date + float vs. integer

Syntax: more editable than JSON,

What is TOML?

Semantics: JSON + date + float vs. integer

Syntax: more editable than JSON, easier to parse than YAML

TOML example

```
toml_stuff = """\  
[project] # Table -> Dictionary  
name = "orbipatch"  
authors = [ # Array -> List  
# Key/Value -> Dictionary  
{ name = "MZ", email = "mz@devskillup.com" }  
]  
"""
```

TOML example

```
import tomli # tomllib in Python 3.11+
import json
print(json.dumps(
    tomli.loads(toml_stuff),
    indent=4,
))
{
  "project": {
    "name": "orbipatch",
    "authors": [
      {
        "name": "MZ",
        "email": "mz@devskillup.com"
      }
    ]
  }
}
```

Originally: Configure build system,

Originally: Configure build system, experimenting with alternatives,

Originally: Configure build system, experimenting with alternatives, setuptools plugins,

Originally: Configure build system, experimenting with alternatives, setuptools plugins, etc.

Originally: Configure build system, experimenting with alternatives, setuptools plugins, etc.

Now:

Originally: Configure build system, experimenting with alternatives, setuptools plugins, etc.

Now: Still that but also:

Originally: Configure build system, experimenting with alternatives, setuptools plugins, etc.

Now: Still that but also: build-system agnostic metadata,

Originally: Configure build system, experimenting with alternatives, setuptools plugins, etc.

Now: Still that but also: build-system agnostic metadata, configure ecosystem tools.

Parsing pyproject.toml

```
parsed = tomli.loads(minimal_pyproject)
print(
    "Build_system_requires:",
    parsed["build-system"].pop("requires")
)
```

```
Build system requires: ['setuptools']
```

Parsing pyproject.toml

```
print(  
    "Build backend",  
    parsed["build-system"].pop("build-backend")  
)
```

```
Build backend setuptools.build_meta
```

Parsing pyproject.toml

```
print(  
    "Project authors:\n",  
    parsed["project"].pop("authors")  
)
```

```
Project authors:  
[{'name': 'MZ', 'email': 'mz@devskillup.com'}]
```

Parsing pyproject.toml

```
print(  
    "Project description:\n",  
    parsed["project"].pop("description"),  
)
```

```
Project description:  
silly project named after a niche math thing
```

Parsing pyproject.toml

```
print(  
    "Project:\n",  
    json.dumps(parsed.pop("project"), indent=2),  
)
```

```
Project:  
{  
    "name": "orbipatch",  
    "version": "2022.3.6.2",  
    "readme": "README.rst"  
}
```

build system

requires:

build system

requires: Dependencies

build system

requires: Dependencies
backend:

build system

requires: Dependencies

backend: Module that has the right methods

project

Must:

project

Must: name, version

project

Must: name, version

Recommended:

project

Must: name, version

Recommended: Short description (usually inline),

project

Must: name, version

Recommended: Short description (usually inline), Long description (usually from file),

project

Must: name, version

Recommended: Short description (usually inline), Long description (usually from file), License (usually from file, can be inlined),

project

Must: name, version

Recommended: Short description (usually inline), Long description (usually from file), License (usually from file, can be inlined), URLs (especially "Homepage")

The tools section

Under "tool.NAME"

```
# [tool.black]
# include = '\.pyi'
print(
    "Black configuration:",
    parsed["tool"]["black"],
)
```

Black configuration: {'include': '\\.pyi'}

Configuring coverage

Like with `setup.cfg`, with prefix `"tool."`:

```
# [tool.coverage.run]
# branch = true
print(
    "Coverage_configuration:",
    parsed["tool"]["coverage"],
)
```

```
Coverage configuration: {'run': {'branch': True}}
```

Configuring isort

Like with setup.cfg, with prefix "tool.":

```
# [tool.isort]
# src_paths = ["isort", "test"]
print(
    "isort_configuration:\n",
    parsed["tool"]["isort"],
)
```

```
isort configuration:
{'src_paths': ['isort', 'test']}
```

project metadata

project section:

project metadata

project section: packaging semantics edition

Configuring dependencies

```
# [project]
# ...
# dependencies = ["six"]
print(
    "dependencies:",
    parsed["project"]["dependencies"],
)
```


Configuring optional dependencies

```
# [project.optional-dependencies]
# tests = ["pytest"]
# docs = ["sphinx"]
print(
    "Optional_dependencies:\n",
    parsed["project"]["optional-dependencies"],
)
```

```
Optional dependencies:
{'tests': ['pytest'], 'docs': ['sphinx']}
```

Configuring console scripts

```
# [project.scripts]
# awesome-command = "my_package:main"
print(
    "scripts:",
    parsed["project"]["scripts"],
)

scripts: {'awesome-command': 'my_package:main'}
```

Configuring entry points

```
# [project.entry-points."paste.app_factory"]  
# main = "my-package:main"  
print(  
    "entry_points:\n",  
    parsed["project"]["entry-points"],  
)  
  
entry_points:  
    {'paste.app_factory': {'main': 'my-package:main'}}
```

Build system

Requires:

Build system

Requires: Usual dependency rules (can include minimal, pinned, etc.)

Build system

Requires: Usual dependency rules (can include minimal, pinned, etc.)

Build system:

Build system

Requires: Usual dependency rules (can include minimal, pinned, etc.)

Build system: PEP 517:

```
def build_wheel(  
    wheel_directory ,  
    config_settings=None ,  
    metadata_directory=None ,  
):  
    ...
```

Packaging Python




```
python -m build
```

Usually:

```
python -m build
```

Usually: Just works

```
python -m build
```

Usually: Just works

Hint:

```
python -m build
```

Usually: Just works

Hint: Use "src/" structure

```
python -m build
```

Usually: Just works

Hint: Use "src/" structure

Sometimes:

```
python -m build
```

Usually: Just works

Hint: Use "src/" structure

Sometimes: BETA!

```
python -m build
```

Usually: Just works

Hint: Use "src/" structure

Sometimes: BETA! "tools.setup.ijsomethingi"

Editable installs

Empty setup.cfg: no longer needed



Dynamic fields

```
# [project]
# name = "orbipatch"
# dynamic = ["version"]
print(
    "name" ,
    parsed[" project" ].pop("name" ) ,
)
print(
    " project" ,
    parsed[" project" ] ,
)

name orbipatch
project { 'dynamic ' : [ 'version ' ] }
```

setuptools scm

```
# [build-system]  
# requires = [  
#     "setuptools",  
#     "setuptools_scm",  
# ]  
#  
# [project]  
# name = "orbipatch"  
# dynamic = ["version"]
```

Packaging!

Packaging!

Also: Everything else

Packaging!

Also: Everything else

Support in your own tooling

project fields

Name

project fields

Name

Version

project fields

Name

Version

Description, license, readme

project fields

Name

Version

Description, license, readme

Dependencies (and Optional dependencies)

project fields

Name

Version

Description, license, readme

Dependencies (and Optional dependencies)

Scripts

project fields

Name

Version

Description, license, readme

Dependencies (and Optional dependencies)

Scripts

Entry points

setuptools support

Defaults usually good

setuptools support

Defaults usually good
Use src/

setuptools support

Defaults usually good

Use src/ (convention over configuration!)

setuptools support

Defaults usually good

Use src/ (convention over configuration!)

Configure lightly where you must

setuptools support

Defaults usually good

Use src/ (convention over configuration!)

Configure lightly where you must

"python -m build" future-proofing