extras-require

Release 0.5.0

Display a warning at the top of module documentation that it has additional requirements.

Dominic Davis-Foster

Apr 24, 2024

Contents

Chapter

ONE

Overview

This extension assumes you have a repository laid out like this:

```
.
├── chemistry_tools
│   ├── __init__.py
│   ├── formulae
│   │   ├── __init__.py
│   │   ├── compound.py
│   │   ├── formula.py
│   │   ├── parser.py
│   │   └── requirements.txt
│   ├── constants.py
│   └── utils.py
├── doc-source
│   ├── api
│   │   ├── chemistry_tools.rst
│   │   ├── elements.rst
│   │   ├── formulae.rst
│   │   └── pubchem.rst
│   ├── conf.py
│   ├── index.rst
│   └── requirements.txt
├── LICENSE
├── README.rst
├── requirements.txt
├── setup.py
└── tox.ini
```

The file `./chemistry_tools/formulae/requirements.txt` contains the additional requirements to run the `formulae` subpackage. These would be defined in `setup.py` like this:

```
setup(
        extras_require={
                "formulae": [
                        "mathematical>=0.1.7",
                        "pandas>=1.0.1",
                        "pyparsing>=2.2.0",
                        "tabulate>=0.8.3",
                        "cawdrey>=0.1.2",
                        "quantities>=0.12.4",
                        ],
                }
        )
```

A message can be displayed in the documentation to indicate that the subpackage has these additional requirements that must be installed.

For instance, this:

```
.. extras-require:: formulae
    :file: formulae/requirements.txt
```

will produce this:

Attention: This module has the following additional requirements:

```
cawdrey>=0.1.2
mathematical>=0.1.7
pandas>=1.0.1
pyparsing>=2.2.0
quantities>=0.12.4
tabulate>=0.8.3
```

These can be installed as follows:

```
$ python -m pip install chemistry_tools[formulae]
```

The path given in `:file:` is relative to the `package_root` variable given in `conf.py`, which in turn is relative to the parent directory of the sphinx documentation. For example, this line:.

```
package_root = "chemistry_tools"
```

points to `./chemistry_tools`, and therefore `:file: formulae/requirements.txt` points to `./chemistry_tools/formulae/requirements.txt`.

Requirements can also be specified in `pyproject.toml` (using the option `:pyproject:`), `setup.cfg` (using the option `:setup.cfg::`), or by typing in the requirements manually, one per line.

The `:scope:` option can be used to specify a different scope for additional requirements, such as `package`, `module`, `class` or `function`. Any string value can be supplied here.

Part I

Usage

Chapter

TWO

Installation

2.1 from PyPI

```
$ python3 -m pip install extras_require --user
```

2.2 from Anaconda

First add the required channels

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

Then install

```
$ conda install extras_require
```

2.3 from GitHub

```
$ python3 -m pip install git+https://github.com/sphinx-toolbox/extras_require@master --user
```

Enable `extras_require` by adding the following to the `extensions` variable in your `conf.py`:

```
extensions = [
    ...
    'sphinx-prompt',
    'sphinxcontrib.extras_require',
    ]
```

For more information see https://www.sphinx-doc.org/en/master/usage/extensions#third-party-extensions.

Chapter

THREE

Configuration

package_root

Type: str
Required: True

Location of package source directory relative to documentation source directory.

For example,

```
package_root = "chemistry_tools"
```

points to ./chemistry_tools.

pypi_name

Type: str
Required: False

The name of the package on PyPI.

This is the name provided to pip install. Defaults to the name of the project in the Sphinx configuration.

New in version 0.4.0.

Chapter

FOUR

Directive

.. extras-require::

The requirements can be specified in several ways:

:file: requirements_file (string)

Shows the requirements from the given file. The file must contain a list of **PEP 508** requirements, one per line.

The path is relative to the `package_root` variable given in `conf.py`, which in turn is relative to the parent directory of the sphinx documentation.

:__pkginfo__: (flag)

Flag to indicate the requirements should be obtained from `__pkginfo__.py`.

This looks in the parent directory of the sphinx documentation for a file named `__pkginfo__.py`. The requirements are imported as the variable `extras_require`, which must be a dictionary mapping extras to a list of requirements.

Example:

```
extras_require = {
        "extra_b": [
                "flask>=1.1.2",
                "click<7.1.2",
                "sphinx==3.0.3",
                ],
        }
```

The requirements can be generated programmatically in the `__pkginfo__.py` file during the import process.

:setup.cfg: (flag)

Flag to indicate the requirements should be obtained from `setup.cfg`.

This looks in the parent directory of the sphinx documentation for a file named `setup.cfg`. This file must be readable by Python's `configparser` module, and contain the section `[options.extras_require]`.

Example:

```
[options.extras_require]
extra_c = faker; pytest; tox
```

See the setuptools documentation for more information on `setup.cfg`.

:flit: (flag)

Flag to indicate the requirements should be obtained from the `[tool.flit.metadata.requires-extra]` section of `pyproject.toml`.

Example:

```
[tool.flit.metadata.requires-extra]
test = [
    "pytest>=2.7.3",
    "pytest-cov",
]
doc = ["sphinx"]
```

See the flit documentation for more details.

:pyproject: (flag)

Flag to indicate the requirements should be obtained from the `[project.optional-dependencies]` section of `pyproject.toml`.

Example:

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

See the **PEP 621** section on dependencies/optional-dependencies for more details.

Only one of the above options can be used in each directive.

Manual requirements:

If none of the above options are provided the **PEP 508** requirements can instead be provided as the content of the directive. Each requirement must be on its own line, and there must be a blank line between the directive and the list of requirements. e.g.

```
.. extras-require:: dates

    pytz >=2019.1
```

Attention: This module has the following additional requirement:

```
pytz>=2019.1
```

This can be installed as follows:

```
$ python -m pip install chemistry_tools[dates]
```

Other options:

:scope: (string)

Specifies a different scope for additional requirements, such as package, module, class or function.

Any string value can be supplied here.

Example

```
.. extras-require:: foo
    :scope: class

    bar
    baz
```

Attention: This class has the following additional requirements:

```
bar
baz
```

These can be installed as follows:

```
$ python -m pip install chemistry_tools[foo]
```

Part II

API Reference

Chapter

FIVE

sphinxcontrib.extras_require

A Sphinx directive to specify that a module has extra requirements, and show how to install them.

Functions:

setup(app)	Setup *sphinxcontrib.extras_require*.

setup(*app*)

Setup *sphinxcontrib.extras_require*.

Parameters **app** (Sphinx) – The Sphinx app.

Return type Dict[str, Any]

Chapter

SIX

sphinxcontrib.extras_require.directive

The `extras-require` directive.

Classes:

`ExtrasRequireDirective`(name, arguments, ...)	Directive to show a notice to users that a module, class or function has additional requirements.

Functions:

`get_requirements`(env, extra, options, content)	Get the requirements for the extras_require node.
`make_node_content`(requirements, ... [, scope])	Create the content of an extras_require node.
`validate_requirements`(requirements_list)	Validate a list of **PEP 508** requirements and format them consistently.

class ExtrasRequireDirective(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

Bases: `SphinxDirective`

Directive to show a notice to users that a module, class or function has additional requirements.

Attributes:

`required_arguments`	One argument is required, the name of the extra (e.g.

Methods:

`run`()	Create the extras_require node.

required_arguments = 1

Type: `int`

One argument is required, the name of the extra (e.g. "testing", "docs")

run()

Create the extras_require node.

Return type `List[Node]`

get_requirements(*env*, *extra*, *options*, *content*)

Get the requirements for the extras_require node.

Parameters

- **env** (`BuildEnvironment`)

- **extra** (str)
- **options** (Dict[str, Any])
- **content** (Union[Iterable, ViewList])

Return type List[str]

make_node_content (*requirements*, *package_name*, *extra*, *scope='module'*)

Create the content of an extras_require node.

Parameters

- **requirements** (List[str]) – List of additional **PEP 508** requirements.
- **package_name** (str) – The name of the module/package on PyPI.
- **extra** (str) – The name of the "extra".
- **scope** (str) – The scope of the additional requirements, e.g. "module", "package". Default 'module'.

Return type str

Returns The content of an extras_require node.

validate_requirements (*requirements_list*)

Validate a list of **PEP 508** requirements and format them consistently.

Parameters **requirements_list** (List[str]) – List of **PEP 508** requirements.

Return type List[str]

Returns List of **PEP 508** requirements with consistent formatting.

Chapter

SEVEN

sphinxcontrib.extras_require.sources

Supported sources for the requirements are implemented here.

Data:

sources	Instance of *Sources*.

Classes:

Sources([iterable])	Class to store functions that provide requirements sources.

Functions:

requirements_from_file(package_root, ...)	Load requirements from the specified file.
requirements_from_flit(package_root, ...)	Load requirements from the `[tool.flit.metadata.requires-extra]` section of a `pyproject.toml` file in the root of the repository.
requirements_from_pkginfo(package_root, ...)	Load requirements from a `__pkginfo__.py` file in the root of the repository.
requirements_from_pyproject(package_root, ...)	Load requirements from the `[project.optional-dependencies]` section of a `pyproject.toml` file in the root of the repository.
requirements_from_setup_cfg(package_root, ...)	Load requirements from a `setup.cfg` file in the root of the repository.

class Sources(*iterable=()*, /)

Bases: `List[Tuple[str, Callable, Callable]]`

Class to store functions that provide requirements sources.

The syntax of each entry is:

```
(option_name, getter_function, validator_function)
```

- a string to use in the directive to specify the source to use,
- the function that returns the list of additional requirements,
- a function to validate the option value provided by the user.

Methods:

register(option_name[, validator])	Decorator to register a function.

register(*option_name*, *validator=<function 'unchanged'>*)

Decorator to register a function.

The function must have the following signature:

```
def function(
    package_root: pathlib.Path,
    options: Dict,
    env: sphinx.environment.BuildEnvironment,
    extra: str,
    ) -> List[str]: ...
```

Parameters

- **option_name** (str) – A string to use in the directive to specify the source to use.
- **validator** (Callable) – A function to validate the option value provided by the user. Default docutils.parsers.rst.directives.unchanged().

Return type Callable

Returns The registered function.

Raises SyntaxError if the decorated function does not take the correct arguments.

requirements_from_file(*package_root*, *options*, *env*, *extra*)

Load requirements from the specified file.

Parameters

- **package_root** (Path) – The path to the package root
- **options** (Dict)
- **env** (BuildEnvironment)
- **extra** (str) – The name of the "extra" that the requirements are for

Return type List[str]

Returns List of requirements

requirements_from_flit(*package_root*, *options*, *env*, *extra*)

Load requirements from the [tool.flit.metadata.requires-extra] section of a pyproject.toml file in the root of the repository.

Parameters

- **package_root** (Path) – The path to the package root.
- **options** (Dict)
- **env** (BuildEnvironment)
- **extra** (str) – The name of the "extra" that the requirements are for.

Return type List[str]

Returns List of requirements.

requirements_from_pkginfo (*package_root*, *options*, *env*, *extra*)
Load requirements from a `__pkginfo__.py` file in the root of the repository.

Parameters

- **package_root** (`Path`) – The path to the package root
- **options** (`Dict`)
- **env** (`BuildEnvironment`)
- **extra** (`str`) – The name of the "extra" that the requirements are for

Return type `List[str]`

Returns List of requirements

requirements_from_pyproject (*package_root*, *options*, *env*, *extra*)
Load requirements from the `[project.optional-dependencies]` section of a `pyproject.toml` file in the root of the repository.

See also: **PEP 621** – Storing project metadata in pyproject.toml

New in version 0.3.0.

Parameters

- **package_root** (`Path`) – The path to the package root.
- **options** (`Dict`)
- **env** (`BuildEnvironment`)
- **extra** (`str`) – The name of the "extra" that the requirements are for.

Return type `List[str]`

Returns List of requirements.

requirements_from_setup_cfg (*package_root*, *options*, *env*, *extra*)
Load requirements from a `setup.cfg` file in the root of the repository.

Parameters

- **package_root** (`Path`) – The path to the package root.
- **options** (`Dict`)
- **env** (`BuildEnvironment`)
- **extra** (`str`) – The name of the "extra" that the requirements are for.

Return type `List[str]`

Returns List of requirements.

sources
Instance of *`Sources`*.

Python Module Index

S

Index