
extras-require

Release 0.5.0

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

Dominic Davis-Foster

May 15, 2024

Contents

1	Overview	1
I	Usage	3
2	Installation	5
2.1	from PyPI	5
2.2	from Anaconda	5
2.3	from GitHub	5
3	Configuration	7
3.1	package_root	7
3.2	pypi_name	7
4	Directive	9
4.1	.. extras-require::	9
II	API Reference	13
5	sphinxcontrib.extras_require	15
5.1	setup	15
6	sphinxcontrib.extras_require.directive	17
6.1	ExtrasRequireDirective	17
6.2	get_requirements	17
6.3	make_node_content	18
6.4	validate_requirements	18
7	sphinxcontrib.extras_require.sources	19
7.1	Sources	19
7.2	requirements_from_file	20
7.3	requirements_from_flit	20
7.4	requirements_from_pkginfo	21
7.5	requirements_from_pyproject	21
7.6	requirements_from_setup_cfg	21
7.7	sources	21
	Python Module Index	23
	Index	25

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

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>.

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.

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

sphinxcontrib.extras_require

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

Functions:

<code>setup(app)</code>	Setup <code>sphinxcontrib.extras_require</code> .
-------------------------	---

setup (*app*)
Setup `sphinxcontrib.extras_require`.

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

Return type `Dict[str, Any]`

`sphinxcontrib.extras_require.directive`

The *extras-require* directive.

Classes:

<code>ExtrasRequireDirective</code> (name, arguments, ...)	Directive to show a notice to users that a module, class or function has additional requirements.
--	---

Functions:

<code>get_requirements</code> (env, extra, options, content)	Get the requirements for the extras_require node.
<code>make_node_content</code> (requirements, ...[, scope])	Create the content of an extras_require node.
<code>validate_requirements</code> (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:

<code>required_arguments</code>	One argument is required, the name of the extra (e.g.
---------------------------------	---

Methods:

<code>run()</code>	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.

sphinxcontrib.extras_require.sources

Supported sources for the requirements are implemented here.

Data:

<code>sources</code>	Instance of <code>Sources</code> .
----------------------	------------------------------------

Classes:

<code>Sources([iterable])</code>	Class to store functions that provide requirements sources.
----------------------------------	---

Functions:

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

<code>register(option_name[, validator])</code>	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

- sphinxcontrib.extras_require, [15](#)
- sphinxcontrib.extras_require.directive,
[17](#)
- sphinxcontrib.extras_require.sources,
[19](#)

Symbols

`:__pkginfo__`: (directive option)
 `extras-require` (directive), 9
`:file`: (directive option)
 `extras-require` (directive), 9
`:flit`: (directive option)
 `extras-require` (directive), 9
`:pyproject`: (directive option)
 `extras-require` (directive), 10
`:scope`: (directive option)
 `extras-require` (directive), 10
`:setup.cfg`: (directive option)
 `extras-require` (directive), 9

E

`extras-require` (directive), 9
 `:__pkginfo__`: (directive option), 9
 `:file`: (directive option), 9
 `:flit`: (directive option), 9
 `:pyproject`: (directive option), 10
 `:scope`: (directive option), 10
 `:setup.cfg`: (directive option), 9
`ExtrasRequireDirective` (class in
 `sphinxcontrib.extras_require.directive`), 17

G

`get_requirements()` (in module
 `sphinxcontrib.extras_require.directive`), 17

M

`make_node_content()` (in module
 `sphinxcontrib.extras_require.directive`), 18
 module
 `sphinxcontrib.extras_require`, 15
 `sphinxcontrib.extras_require.directive`,
 17
 `sphinxcontrib.extras_require.sources`,
 19

P

`package_root` (configuration value), 7
`pypi_name` (configuration value), 7
 Python Enhancement Proposals

PEP 508, 9, 10, 17, 18
 PEP 621, 10, 21

R

`register()` (*Sources* method), 20
`required_arguments` (*ExtrasRequireDirective*
 attribute), 17
`requirements_from_file()` (in module
 `sphinxcontrib.extras_require.sources`), 20
`requirements_from_flit()` (in module
 `sphinxcontrib.extras_require.sources`), 20
`requirements_from_pkginfo()` (in module
 `sphinxcontrib.extras_require.sources`), 21
`requirements_from_pyproject()` (in module
 `sphinxcontrib.extras_require.sources`), 21
`requirements_from_setup_cfg()` (in module
 `sphinxcontrib.extras_require.sources`), 21
`run()` (*ExtrasRequireDirective* method), 17

S

`setup()` (in module `sphinxcontrib.extras_require`), 15
`Sources` (class in
 `sphinxcontrib.extras_require.sources`), 19
`sources` (in module
 `sphinxcontrib.extras_require.sources`), 21
`sphinxcontrib.extras_require`
 module, 15
`sphinxcontrib.extras_require.directive`
 module, 17
`sphinxcontrib.extras_require.sources`
 module, 19

V

`validate_requirements()` (in module
 `sphinxcontrib.extras_require.directive`), 18