sphinx-packaging

Release 0.2.0

A collection of Sphinx utilities related to Python packaging.

Dominic Davis-Foster

May 15, 2024

Contents

Chapter

ONE

Installation

1.1 from PyPI

```
$ python3 -m pip install sphinx-packaging --user
```

1.2 from GitHub

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

Chapter

TWO

Usage

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

```
extensions = [
    ...
    'sphinx_packaging',
    ]
```

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

2.1 Directives

`.. tconf:: name`
Used to document a TOML configuration field, for example in **PEP 621** metadata.

:type: (string)
Indicates the field's type.

:required: (flag)
Indicates the whether a value is required for the field.

:default: (string)
Indicates the default value for the field.

:noindex: (flag)
Disables the index entry and cross-referencing for this field.

Examples:

```
.. tconf:: project.name
    :type: :toml:`String`
    :required: True

    The name of the project.

.. tconf:: project.version
    :type: :toml:`String`
    :required: True

    The version of the project as supported by :pep:`440`.

.. tconf:: description
```

(continues on next page)

(continued from previous page)

```
    :type: :toml:`String`
    :default: ``'This is the description'``
    :required: False

    A short summary description of the project.
```

`project.`**`name`**
: **Type:** String
Required: `True`

  The name of the project.

`project.`**`version`**
: **Type:** String
Required: `True`

  The version of the project as supported by **PEP 440**.

`description`
: **Type:** String
Required: `False`
Default: `'This is the description'`

  A short summary description of the project.

2.2 Roles

`sphinx_packaging` provides the following roles:

`:pep:`
: Creates a cross-reference to a Python Enhancement Proposal.

  The text "PEP NNN" is inserted into the document, and with supported builders is a hyperlink to the online copy of the specified PEP. This role also generates an appropriate index entry.

  You can link to a specific section by writing `:pep:`number#anchor``.

  A custom title can also be added to the link by writing `:pep:`title <number#anchor>``. Unlike the version of this directive which ships with Sphinx, the link is not shown in bold when there is a custom title.

  Examples:

```
:pep:`621`

:pep:`503#normalized-names`

.. seealso:: The :pep:`specification <427>` for wheels.
```

  PEP 621

  PEP 503#normalized-names

  See also:

  The specification for wheels.

:pep621:

Creates a cross-reference to a section in **PEP 621**, typically the name of a field in `pyproject.toml`.

The title of the directive (either implicit, `:pep621:`title``, or explicit `:pep621:`title <target>``) is inserted into the document. With supported builders is a hyperlink to the specified heading in the online copy of the PEP. This role also generates an appropriate index entry.

Examples:

```
The :pep621:`name` field must be provided and cannot be :pep621:`dynamic`.

:pep621:`Version <version>` may be required by some backend,
but can be determined dynamically by others.

:pep621:`authors` and :pep621:`maintainers` both point to the same section.
```

The name field must be provided and cannot be dynamic.

Version may be required by some backend, but can be determined dynamically by others.

authors and maintainers both point to the same section.

:core-meta:

Creates a cross-reference to a field in the Python core metadata.

The title of the directive (either implicit, `:core-meta:`title``, or explicit `:core-meta:`title <target>``) is inserted into the document. With supported builders is a hyperlink to the specified field in the specification on packaging.python.org. This role also generates an appropriate index entry.

Examples:

```
:core-meta:`Supported-Platform (Multiple Use) <Supported-Platform>` specifies the␣
↪OS and CPU
for which the binary distribution was compiled.

The project's :core-meta:`description` may have multiple lines.

:pep621:`requires-python` in ``pyproject.toml`` maps to :core-meta:`Requires-
↪Python`
```

Supported-Platform (Multiple Use) specifies the OS and CPU for which the binary distribution was compiled.

The project's description may have multiple lines.

requires-python in `pyproject.toml` maps to Requires-Python

:toml:

Creates a cross-reference to a section in the TOML specification.

The title of the directive (either implicit, `:toml:`title``, or explicit `:toml:`title <target>``) is inserted into the document. With supported builders is a hyperlink to section in the web version of the specification. This role also generates an appropriate index entry.

Examples:

```
TOML's :toml:`string` type accepts either single or double quotes.

:toml:`Inline Tables <Inline Table>` must be on a single line.
```

(continues on next page)

(continued from previous page)

```
There are four date/time types in TOML:

* :toml:`Offset Date-Time`
* :toml:`Local Date-Time`
* :toml:`Local Date`
* :toml:`!Local Time`
```

TOML's string type accepts either single or double quotes.

Inline Tables must be on a single line.

There are four date/time types in TOML:

- Offset Date-Time
- Local Date-Time
- Local Date
- Local Time

The last xref will not appear in the index because the target is prefixed with a `!`. This also works when there is an explicit title:

```
The following xrefs are not indexed: :toml:`!Float`, :toml:`array <!Array>`.
```

The following xrefs are not indexed: Float, array.

:tconf:

Role which provides a cross-reference to a *tconf* directive.

Examples:

```
:tconf:`project.name` and :tconf:`~project.version` are required.
Some backends may be able to determine a value for :tconf:`version` dynamically.

:tconf:`description` will be displayed this towards the top of the project page␣
↪on PyPI.

Links can also be written with a shorter name: :tconf:`~.name`.
```

project.name and *version* are required. Some backends may be able to determine a value for *version* dynamically.

description will be displayed this towards the top of the project page on PyPI.

Links can also be written with a shorter name: *name*.

2.3 Configuration

toml_spec_version

Type: string
Required: `False`
Default: 1.0.0

The version of the TOML specification to link to.

For example, this documentation links to `v0.5.0` with the following setting:

```
# conf.py
toml_spec_version = "0.5.0"
```

tconf_show_full_name

Type: `bool`
Required: `False`
Default: True

Whether to show the full name for field.

For example, with `tconf_show_full_name = True`:

`project.`**description**

Type: String

And with `tconf_show_full_name = False`:

description

Type: String

Chapter

THREE

sphinx_packaging

A collection of Sphinx utilities related to Python packaging.

Functions:

setup(app)	Setup *sphinx_packaging*.

setup(*app*)

Setup *sphinx_packaging*.

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

Return type Dict[str, Any]

Chapter

FOUR

sphinx_packaging.peps

Sphinx extension which modifies the `pep` role to use normal (i.e. not bold) text for custom titles.

Also adds the *pep621* role for referencing sections within **PEP 621**, and the *core-meta* role for referencing sections in Python's core metadata`.

Classes:

CoreMetadata()	Sphinx role for referencing a core metadata field.
PEP()	Sphinx role for referencing a PEP or a section thereof.
PEP621Section()	Sphinx role for referencing a section within **PEP 621**.

Functions:

setup(app)	Setup *sphinx_packaging.peps*.

class CoreMetadata

Bases: `ReferenceRole`

Sphinx role for referencing a core metadata field.

Methods:

build_uri()	Construct the target URI for the reference node.
run()	Process the role.

build_uri()

Construct the target URI for the reference node.

Return type `str`

run()

Process the role.

Return type `Tuple`[`List`[`Node`], `List`[`system_message`]]

class PEP

Bases: ReferenceRole

Sphinx role for referencing a PEP or a section thereof.

Methods:

build_uri()	Constrict the target URI for the reference node.
run()	Process the role.

build_uri()

Constrict the target URI for the reference node.

Return type str

run()

Process the role.

Return type Tuple[List[Node], List[system_message]]

class PEP621Section

Bases: *PEP*

Sphinx role for referencing a section within **PEP 621**.

Methods:

__call__(name, rawtext, text, lineno, inliner)	Call self as a function.

__call__(*name*, *rawtext*, *text*, *lineno*, *inliner*, *options={}*, *content=[]*)

Call self as a function.

Return type Tuple[List[Node], List[system_message]]

setup(*app*)

Setup *sphinx_packaging.peps*.

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

Return type Dict[str, Any]

Chapter

FIVE

sphinx_packaging.tconf

The *tconf* directive and role for configuration fields in `pyproject.toml` etc.

Classes:

TConfXRefRole([fix_parens, lowercase, ...])	Customised XRef role for *tconf* roles.
TOMLConf(name, arguments, options, content, ...)	The *tconf* directive.

Functions:

resolve_xref(app, env, node, contnode)	Resolve as-yet-unresolved XRefs for *tconf* roles.
setup(app)	Setup *sphinx_packaging.tconf*.

class TConfXRefRole (*fix_parens=False*, *lowercase=False*, *nodeclass=None*, *innernodeclass=None*, *warn_dangling=False*)

Bases: `XRefRole`

Customised XRef role for *tconf* roles.

Methods:

process_link(env, refnode, ...)	Construct a link from the parsed content of the role.

process_link (*env*, *refnode*, *has_explicit_title*, *title*, *target*)

Construct a link from the parsed content of the role.

Parameters

- **env** (`BuildEnvironment`) – The Sphinx build environment.
- **refnode** (`Element`) – The reference node.
- **has_explicit_title** (`bool`) – Whether the role has an explicit title.
- **title** (`str`) – The title of the XRef role.
- **target** (`str`) – The target of the XRef role. (`:tconf:`title <target>``)

Return type `Tuple`[`str`, `str`]

Returns A tuple of `(title, target)`.

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

Bases: GenericObject

The *tconf* directive.

Methods:

format_default(default)	Formats the :default: option.
format_required(required)	Formats the :required: option.
format_type(the_type)	Formats the :type: option.
handle_signature(sig, signode)	Parse the signature of the *tconf* directive.
run()	Process the content of the directive.

Attributes:

indextemplate	The template string for index entries.

static format_default(*default*)

Formats the :default: option.

Parameters **default** (str)

Return type str

static format_required(*required*)

Formats the :required: option.

Parameters **required** (str)

Return type bool

static format_type(*the_type*)

Formats the :type: option.

Parameters **the_type** (str)

Return type str

handle_signature(*sig*, *signode*)

Parse the signature of the *tconf* directive.

Parameters

- **sig** (str) – The name of the field.
- **signode** (desc_signature) – The signature node created by Sphinx.

Return type str

Returns The final component of the field path (e.g. foo.bar -> bar).

indextemplate = 'pair: %s; TOML configuration field'

Type: str

The template string for index entries.

run()
Process the content of the directive.

Return type List[Node]

resolve_xref(*app*, *env*, *node*, *contnode*)
Resolve as-yet-unresolved XRefs for *tconf* roles.

Parameters

- **app** (Sphinx) – The Sphinx application.
- **env** (BuildEnvironment) – The Sphinx build environment.
- **node** (Node) – The cross reference node which has not yet been.
- **contnode** (Node) – The child node of the reference node, which provides the formatted text.

Return type Optional[reference]

setup(*app*)
Setup *sphinx_packaging.tconf*.

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

Return type Dict[str, Any]

Chapter

SIX

sphinx_packaging.toml

Sphinx extension which adds the *toml* role for referencing sections of the TOML specification.

Classes:

TOML()	Sphinx role for referencing a section of the TOML specification.

Functions:

setup(app)	Setup *sphinx_packaging.toml*.

class TOML

Bases: ReferenceRole

Sphinx role for referencing a section of the TOML specification.

Methods:

build_uri()	Constrict the target URI for the reference node.
run()	Process the role.

build_uri()

Constrict the target URI for the reference node.

Return type str

run()

Process the role.

Return type Tuple[List[Node], List[system_message]]

setup(*app*)

Setup *sphinx_packaging.toml*.

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

Return type Dict[str, Any]

Chapter

SEVEN

Downloading source code

The `sphinx-packaging` source code is available on GitHub, and can be accessed from the following URL: https://github.com/sphinx-toolbox/sphinx-packaging

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/sphinx-toolbox/sphinx-packaging
```

```
Cloning into 'sphinx-packaging'...
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a 'zip' file by clicking:
Clone or download –> Download Zip

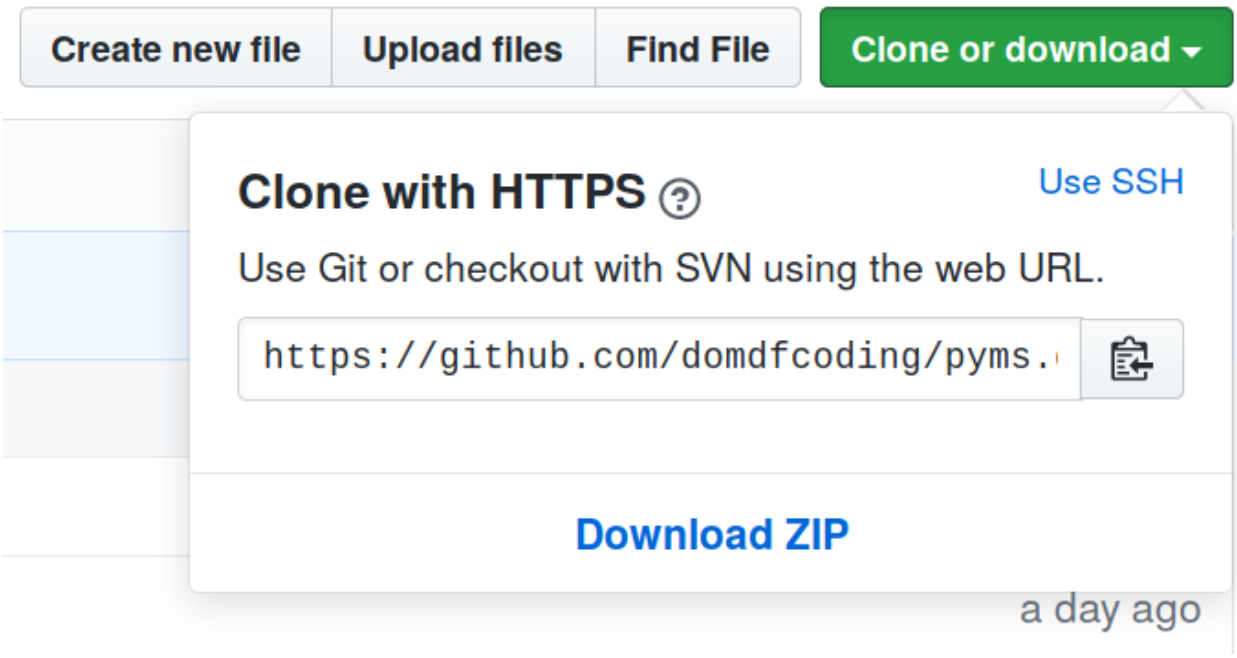

Fig. 1: Downloading a 'zip' file of the source code

7.1 Building from source

The recommended way to build `sphinx-packaging` is to use tox:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use pep517.build or another **PEP 517**-compatible build tool.

Chapter

EIGHT

License

`sphinx-packaging` is licensed under the BSD 3-Clause "New" or "Revised" License

---

A permissive license similar to the BSD 2-Clause License, but with a 3rd clause that prohibits others from using the name of the copyright holder or its contributors to promote derived products without written consent.

Permissions

- Commercial use – The licensed material and derivatives may be used for commercial purposes.
- Modification – The licensed material may be modified.
- Distribution – The licensed material may be distributed.
- Private use – The licensed material may be used and modified in private.

Conditions

- License and copyright notice – A copy of the license and copyright notice must be included with the licensed material.

Limitations

- Liability – This license includes a limitation of liability.
- Warranty – This license explicitly states that it does NOT provide any warranty.

See more information on choosealicense.com ⇒

```
Copyright (c) 2021, Dominic Davis-Foster

All rights reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright notice,
      this list of conditions and the following disclaimer in the documentation
      and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
```

(continues on next page)

(continued from previous page)

```
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

Python Module Index

S

Index

Symbols

B

C

D

F

H

I

M

P

R

S

T