enum-tools

Release 0.12.0

Tools to expand Python's enum module.

Dominic Davis-Foster

May 15, 2024

Contents

Chapter

ONE

Overview

This library provides the following:

1. *enum_tools.autoenum* – A Sphinx extension to document Enums better than `autoclass` can currently.
2. *@enum_tools.documentation.document_enum* – A decorator to add docstrings to `Enum` members from a comment at the end of the line.
3. *enum_tools.custom_enums* – Additional `Enum` classes with different functionality.

Chapter

TWO

Installation

2.1 from PyPI

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

2.2 from Anaconda

First add the required channels

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

Then install

```
$ conda install enum_tools
```

2.3 from GitHub

```
$ python3 -m pip install git+https://github.com/domdfcoding/enum_tools@master --user
```

Chapter

THREE

`enum_tools.autoenum` – Sphinx Extension

A Sphinx directive for documenting `Enums` in Python.

Provides the *`autoenum`* directive for documenting `Enums`, and *`autoflag`* for documenting `Flags`. These behaves much like `autoclass` and `autofunction`.

Attention: This extension / module has the following additional requirements:

```
sphinx>=3.4.0
sphinx-jinja2-compat>=0.1.1
sphinx-toolbox>=2.16.0
```

These can be installed as follows:

```
$ python -m pip install enum-tools[sphinx]
```

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

```
extensions = [
    ...
    'enum_tools.autoenum',
    ]
```

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

Sections

3.1 Usage

`.. autoenum::`
`.. autoflag::`

These directives are used for documenting `Enums` and `Flags` respectively.

They support the same options as `autoclass`, but with a few changes to the behaviour:

- Enum members are always shown regardless of whether they are documented or not.
- Enum members are grouped separately from methods.

The docstrings of the Enum members are taken from their `__doc__` attributes. This can be set during initialisation of the enum (see an example here), with the *DocumentedEnum* class, or with the *document_enum()* decorator.

See the autodoc module documentation for further details of the general `autoclass` behaviour.

:py:enum:mem:
:py:enum:member:
:py:flag:mem:
:py:flag:member:

These roles provide cross-references to Enum/Flag members.

New in version 0.4.0.

Unlike a standard `:class:` or `:enum:` xref the default behaviour of the ~ prefix is to show both the Enum's name and the member's name. For example:

```
:py:enum:mem:`~enum_tools.demo.StatusFlags.Running`
```

StatusFlags.Running

The original behaviour can be restored by using the + prefix:

```
:py:enum:mem:`+enum_tools.demo.StatusFlags.Running`
```

Running

3.2 Demo

These two have been created with `automodule`.

```
.. automodule:: enum_tools.demo
    :members:
```

enum NoMethods(*value*)

Bases: `enum.IntEnum`

An enumeration of people without any methods.

Member Type `int`

Valid values are as follows:

Bob = <NoMethods.Bob: 1>
A person called Bob

Alice = <NoMethods.Alice: 2>
A person called Alice

Carol = <NoMethods.Carol: 3>
A person called Carol

enum People(*value*)

Bases: enum.IntEnum

An enumeration of people.

Member Type int

Valid values are as follows:

Bob = <People.Bob: 1>

A person called Bob

Alice = <People.Alice: 2>

A person called Alice

Carol = <People.Carol: 3>

A person called Carol.

This is a multiline docstring.

Dennis = <People.Dennis: 4>

A person called Dennis

The Enum and its members also have the following methods:

classmethod iter_values()

Iterate over the values of the Enum.

classmethod as_list()

Return the Enum's members as a list.

Return type List

This one has been created with autoenum.

```
.. autoenum:: enum_tools.demo.People
    :members:
```

enum People(*value*)

Bases: enum.IntEnum

An enumeration of people.

Member Type int

Valid values are as follows:

Bob = <People.Bob: 1>

A person called Bob

Alice = <People.Alice: 2>

A person called Alice

Carol = <People.Carol: 3>

A person called Carol.

This is a multiline docstring.

Dennis = <People.Dennis: 4>
A person called Dennis

The Enum and its members also have the following methods:

classmethod iter_values()
Iterate over the values of the Enum.

classmethod as_list()
Return the Enum's members as a list.

Return type List

If members don't have their own docstrings no docstring is shown:

```
.. autoenum:: enum_tools.demo.NoMemberDoc
    :members:
```

enum NoMemberDoc(*value*)

Bases: enum.IntEnum

An enumeration of people without any member docstrings.

Member Type int

Valid values are as follows:

Bob = <NoMemberDoc.Bob: 1>

Alice = <NoMemberDoc.Alice: 2>

Carol = <NoMemberDoc.Carol: 3>

Flags can also be documented:

```
.. autoflag:: enum_tools.demo.StatusFlags
    :members:
```

flag StatusFlags(*value*)

Bases: enum.IntFlag

An enumeration of status codes.

Member Type int

Valid values are as follows:

Running = <StatusFlags.Running: 1>
The system is running.

Stopped = <StatusFlags.Stopped: 2>
The system has stopped.

Error = <StatusFlags.Error: 4>
An error has occurred.

The Flag and its members also have the following methods:

has_errored()
Returns whether the operation has errored.

Return type bool

3.3 API Reference

Classes:

EnumDocumenter(*args)	Sphinx autodoc `Documenter` for documenting `Enum`s.
EnumMemberDocumenter(directive, name[, indent])	Sphinx autodoc `Documenter` for documenting `Enum` members.
FlagDocumenter(*args)	Sphinx autodoc `Documenter` for documenting `Flag`s.
PyEnumXRefRole([fix_parens, lowercase, ...])	XRefRole for Enum/Flag members.

Functions:

setup(app)	Setup Sphinx Extension.

class EnumDocumenter(**args*)

Bases: `ClassDocumenter`

Sphinx autodoc `Documenter` for documenting `Enum`s.

Methods:

can_document_member(member, membername, ...)	Called to see if a member can be documented by this documenter.
document_members([all_members])	Generate reST for member documentation.
generate([more_content, real_modname, ...])	Generate reST for the object given by *self.name*, and possibly for its members.

classmethod can_document_member(*member*, *membername*, *isattr*, *parent*)

Called to see if a member can be documented by this documenter.

Parameters

- **member** (`Any`)
- **membername** (`str`)
- **isattr** (`bool`)
- **parent** (`Any`)

Return type `bool`

document_members(*all_members=False*)

Generate reST for member documentation.

Parameters **all_members** (`bool`) – If `True`, document all members, otherwise document those given by `self.options.members`. Default `False`.

generate(*more_content=None*, *real_modname=None*, *check_module=False*, *all_members=False*)
Generate reST for the object given by *self.name*, and possibly for its members.

Parameters

- **more_content** (Optional[Any]) – Additional content to include in the reST output. Default None.
- **real_modname** (Optional[str]) – Module name to use to find attribute documentation. Default None.
- **check_module** (bool) – If True, only generate if the object is defined in the module name it is imported from. Default False.
- **all_members** (bool) – If True, document all members. Default False.

class EnumMemberDocumenter(*directive*, *name*, *indent=''*)
Bases: AttributeDocumenter

Sphinx autodoc Documenter for documenting Enum members.

Methods:

add_directive_header(sig)	Add the directive header for the Enum member.
generate([more_content, real_modname, ...])	Generate reST for the object given by self.name, and possibly for its members.
import_object([raiseerror])	Import the object given by self.modname and self.objpath and set it as self.object.

add_directive_header(*sig*)
Add the directive header for the Enum member.

Parameters **sig** (str)

generate(*more_content=None*, *real_modname=None*, *check_module=False*, *all_members=False*)
Generate reST for the object given by self.name, and possibly for its members.

Parameters

- **more_content** (Optional[Any]) – Additional content to include in the reST output. Default None.
- **real_modname** (Optional[str]) – Module name to use to find attribute documentation. Default None.
- **check_module** (bool) – If True, only generate if the object is defined in the module name it is imported from. Default False.
- **all_members** (bool) – If True, document all members. Default False.

Changed in version 0.8.0: Multiline docstrings are now correctly represented in the generated output.

import_object(*raiseerror=False*)
Import the object given by self.modname and self.objpath and set it as self.object.

Parameters **raiseerror** (bool) – Default False.

Return type bool

Returns True if successful, False if an error occurred.

class FlagDocumenter(**args*)

Bases: *EnumDocumenter*

Sphinx autodoc `Documenter` for documenting `Flag`s.

Methods:

can_document_member(member, membername, ...)	Called to see if a member can be documented by this documenter.

classmethod can_document_member(*member*, *membername*, *isattr*, *parent*)

Called to see if a member can be documented by this documenter.

Parameters

- **member** (`Any`)
- **membername** (`str`)
- **isattr** (`bool`)
- **parent** (`Any`)

Return type `bool`

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

Bases: `PyXRefRole`

XRefRole for Enum/Flag members.

New in version 0.4.0.

Methods:

process_link(env, refnode, ...)	Called after parsing title and target text, and creating the reference node (given in `refnode`).

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

Called after parsing title and target text, and creating the reference node (given in `refnode`).

This method can alter the reference node and must return a new (or the same) `(title, target)` tuple.

Parameters

- **env** (`BuildEnvironment`)
- **refnode** (`Element`)
- **has_explicit_title** (`bool`)
- **title** (`str`)
- **target** (`str`)

Return type `Tuple[str, str]`

setup(*app*)
Setup Sphinx Extension.

Parameters **app** (Sphinx)

Return type Dict[str, Any]

Chapter

FOUR

enum_tools.custom_enums

Custom subclasses of `enum.Enum` and `enum.Flag`.

Classes:

AutoNumberEnum(value)	`Enum` that automatically assigns increasing values to members.
DuplicateFreeEnum(*args)	`Enum` that disallows duplicated member names.
IntEnum(value)	`Enum` where members are also (and must be) ints.
IterableFlag(value)	`Flag` with support for iterating over members and member combinations.
IterableIntFlag(value)	`IntFlag` with support for iterating over members and member combinations.
MemberDirEnum(value)	`Enum` which includes attributes as well as methods.
OrderedEnum(value)	`Enum` that adds ordering based on the values of its members.
StrEnum(value)	`Enum` where members are also (and must be) strings.

enum AutoNumberEnum(*value*)

Bases: `enum.Enum`

`Enum` that automatically assigns increasing values to members.

enum DuplicateFreeEnum(*value*)

Bases: `enum.Enum`

`Enum` that disallows duplicated member names.

enum IntEnum(*value*)

Bases: `int`, `enum.Enum`

`Enum` where members are also (and must be) ints.

Member Type `int`

flag IterableFlag(*value*)

Bases: `enum.Flag`

`Flag` with support for iterating over members and member combinations.

This functionality was added to Python 3.10's `enum` module in python/cpython#22221.

New in version 0.5.0.

The `Flag` and its members have the following methods:

__iter__()

Returns members in definition order.

Return type `Iterator`[`Flag`]

flag IterableIntFlag(*value*)

Bases: enum.IntFlag

IntFlag with support for iterating over members and member combinations.

This functionality was added to Python 3.10's enum module in python/cpython#22221.

New in version 0.5.0.

Member Type int

The Flag and its members have the following methods:

__iter__()

Returns members in definition order.

Return type Iterator[IntFlag]

enum MemberDirEnum(*value*)

Bases: enum.Enum

Enum which includes attributes as well as methods.

This will be part of the enum module starting with Python 3.10.

See also: Pull request python/cpython#19219 by Angelin BOOZ, which added this to CPython.

New in version 0.6.0.

enum OrderedEnum(*value*)

Bases: enum.Enum

Enum that adds ordering based on the values of its members.

The Enum and its members have the following methods:

__ge__(*other*)

Return self>=value.

Return type bool

__gt__(*other*)

Return self>value.

Return type bool

__le__(*other*)

Return self<=value.

Return type bool

__lt__(*other*)

Return self<value.

Return type bool

enum StrEnum(*value*)

Bases: str, enum.Enum

Enum where members are also (and must be) strings.

Member Type `str`

Chapter

FIVE

enum_tools.documentation

5.1 Core Functionality

Decorators to add docstrings to enum members from comments.

Classes:

DocumentedEnum(value)	An enum where docstrings are automatically added to members from comments starting with `doc:`.

Functions:

document_enum(an_enum)	Document all members of an enum by parsing a docstring from the Python source..
document_member(enum_member)	Document a member of an enum by adding a comment to the end of the line that starts with `doc:`.

enum DocumentedEnum(*value*)

Bases: `enum.Enum`

An enum where docstrings are automatically added to members from comments starting with `doc:`.

Note: This class does not (yet) support the other docstring formats *@document_enum* does.

@document_enum(*an_enum*)

Document all members of an enum by parsing a docstring from the Python source..

The docstring can be added in several ways:

1. A comment at the end the line, starting with `doc:`:

```
Running = 1  # doc: The system is running.
```

2. A comment on the previous line, starting with `#:`. This is the format used by Sphinx.

```
#: The system is running.
Running = 1
```

3. A string on the line *after* the attribute. This can be used for multiline docstrings.

```
Running = 1
"""
The system is running.
```

(continues on next page)

(continued from previous page)

```
Hello World
"""
```

If more than one docstring format is found for an enum member a *MultipleDocstringsWarning* is emitted.

Parameters **an_enum** (enum.Enum) – An Enum subclass

Returns The same object passed as an_enum. This allows this function to be used as a decorator.

Return type enum.Enum

Changed in version 0.8.0: Added support for other docstring formats and multiline docstrings.

document_member(*enum_member*)

Document a member of an enum by adding a comment to the end of the line that starts with doc:.

Parameters **enum_member** (Enum) – A member of an Enum subclass

5.2 Utilities

Exceptions:

MultipleDocstringsWarning(member, docstrings])	Warning emitted when multiple docstrings are found for a single Enum member.

Functions:

get_base_indent(base_indent, all_tokens, indent)	Determine the base level of indentation (i.e.
get_dedented_line(line)	Returns the line without indentation, and the amount of indentation.
get_tokens(line)	Returns a list ot tokens generated from the given Python code.
parse_tokens(all_tokens)	Parse the tokens representing a line of code to identify Enum members and `doc:` comments.

get_base_indent(*base_indent*, *all_tokens*, *indent*)

Determine the base level of indentation (i.e. one level of indentation in from the `c` of `class`).

Parameters

- **base_indent** (Optional[int]) – The current base level of indentation
- **all_tokens** (Sequence[Sequence])
- **indent** (int) – The current level of indentation

Return type Optional[int]

Returns The base level of indentation

get_dedented_line(*line*)

Returns the line without indentation, and the amount of indentation.

Parameters **line** (str) – A line of Python source code

Return type Tuple[int, str]

get_tokens(*line*)

Returns a list ot tokens generated from the given Python code.

Parameters **line** (str) – Line of Python code to tokenise.

Return type List[Tuple]

parse_tokens(*all_tokens*)

Parse the tokens representing a line of code to identify Enum members and `doc:` comments.

Parameters **all_tokens**

Returns A list of the Enum members' names, and the docstring for them.

5.3 Warnings

exception MultipleDocstringsWarning(*member*, *docstrings=()*)

Bases: UserWarning

Warning emitted when multiple docstrings are found for a single Enum member.

New in version 0.8.0.

Parameters

- **member** (Enum)
- **docstrings** (Iterable[str]) – The list of docstrings found for the member. Default ().

__str__()

Return str(self).

Return type str

docstrings

Type: Iterable[str]

The list of docstrings found for the member.

member

Type: Enum

The member with multiple docstrings.

Chapter

SIX

enum_tools.utils

General utility functions.

Classes:

HasMRO	`typing.Protocol` for classes that have a method resolution order magic method (`__mro__`).

Functions:

get_base_object(enum)	Returns the object type of the enum's members.
is_enum(obj)	Returns `True` if `obj` is an `enum.Enum`.
is_enum_member(obj)	Returns `True` if `obj` is an `enum.Enum` member.
is_flag(obj)	Returns `True` if `obj` is an `enum.Flag`.

protocol HasMRO

Bases: `Protocol`

`typing.Protocol` for classes that have a method resolution order magic method (`__mro__`).

This protocol is runtime checkable.

Classes that implement this protocol must have the following methods / attributes:

__mro__ = (<class 'enum_tools.utils.HasMRO'>, <class 'typing_extensions.Protocol'>, <c

Type: `tuple`

__non_callable_proto_members__ = {'__mro__'}

Type: `set`

get_base_object(*enum*)

Returns the object type of the enum's members.

If the members are of indeterminate type then the `object` class is returned.

Parameters **enum** (`Type`[*HasMRO*])

Return type `Type`

Raises **TypeError** – If `enum` is not an Enum.

is_enum(*obj*)

Returns `True` if `obj` is an `enum.Enum`.

Parameters **obj** (`Type`)

Return type `bool`

is_enum_member(*obj*)
Returns `True` if `obj` is an `enum.Enum` member.

Parameters **obj** (`Type`)

Return type `bool`

is_flag(*obj*)
Returns `True` if `obj` is an `enum.Flag`.

Parameters **obj** (`Type`)

Return type `bool`

Python Module Index

e

Index

Symbols

A

B

C

D

E

F

G

H

I

M

P

R

S