autodocsumm Documentation

Release 0.1.13

Philipp Sommer

Mar 05, 2020

Contents

Welcome! This sphinx extension provides some useful extensions to the Sphinxs autodoc extension. Those are

1. It creates a *Table of Contents* in the style of the autosummary extension with methods, classes, functions and attributes
2. As you can include the `__init__` method documentation for via the `autoclass_content` configuration value, we provide the *`autodata_content`* configuration value to include the documentation from the `__call__` method
3. You can exclude the string representation of specific objects. E.g. if you have a large dictionary using the *`not_document_data`* configuration value.

See the *Examples* section for more details.

CHAPTER 1

Content

1.1 Configuration of the sphinx extension

The module provides 3 additional configuration values, one event and new flags for the autodoc directives `autoclass` and `automodule`.

1.1.1 Additional flags for autodoc directives

The most important new flag for the `autoclass` and `automodule` directives is the `autosummary` flag. If you want to have an automatically generated summary to your class or module, you have to add this flag, e.g. via:

```
.. autodoc:: MyClass
    :autosummary:
```

or in the `autodoc_default_options` configuration value of sphinx via

```
autodoc_default_options = {'autosummary': True}
```

See also the usage in the *Examples* section.

The other additional flags lets you control what should be in the autosummary table: these are

- `autosummary-private-members`
- `autosummary-undoc-members`
- `autosummary-inherited-members`
- `autosummary-special-members`
- `autosummary-exlude-members`
- `autosummary-imported-members`
- `autosummary-ignore-module-all`
- `autosummary-members`

- `autosummary-no-nesting`

They are essentially the same as the options for `autodoc` (i.e. `private-members` or `members`), but they only affect the autosummary table (see the example in *Generating a summary table without the full documentation*).

The new additional flag `autosummary-no-nesting` only generates autosummary tables for a module, but not for members within. See *Generating a summary table for the module without nesting*.

1.1.2 Configuration values and events

autodocsumm-grouper (*app*, *what*, *name*, *obj*, *section*, *parent*)

Emitted when autodocsumm has to determine the section for a member in the table of contents. If the return value is None, the given *section* will be used.

Parameters

- **app** – the `Sphinx` application object
- **what** – the type of the object which the docstring belongs to (one of `"module"`, `"class"`, `"exception"`, `"function"`, `"method"`, `"attribute"`)
- **name** – the fully qualified name of the object
- **obj** – the member object
- **section** – The section title that would be given to the object automatically (one of `"Classes"`, `"Exceptions"`, `"Functions"`, `"Methods"`, `"Attributes"`, `"Data"`)
- **parent** – The parent object holding the member *obj*

autodata_content

As you can include the `__init__` method documentation for via the `autoclass_content` configuration value, this configuration value lets you include the documentation from the `__call__` method. Possible values are

class To only use the class documentation

call To only use the documentation from the `__call__` method

both To use the documentation from all.

The default is `'call'`

document_data

To include the string representation of specific data objects. You may provide a list of fully qualified object names (e.g. in the form of `'zipfile.ZipFile'`) or `True` or `False`

not_document_data

To exclude the string representation of specific data objects. You may provide a list of fully qualified object names (e.g. in the form of `'zipfile.ZipFile'`) or `True` or `False`

1.2 Examples

1.2.1 Some module

Just a dummy module with some class definition

Classes

`MyClass`	Some class

Data

`large_data`	Some module data

class MyClass
Some class

With some description

Methods

`do_something`()	Do something

Attributes

`some_attr`	Any instance attribute
`some_other_attr`	Any other instance attribute

do_something()
Do something

some_attr = None
Any instance attribute

some_other_attr = None
Any other instance attribute

large_data = 'Whatever'
Some module data

1.2.2 Demo Class

class MyClass
Some class

With some description

Methods

`do_something`()	Do something

Attributes

`some_attr`	Any instance attribute
`some_other_attr`	Any other instance attribute

do_something()
Do something

some_attr = None
Any instance attribute

some_other_attr = None
Any other instance attribute

1.2.3 Demo Grouper

class MyClass
Some class

With some description

Methods

do_something()	Do something

Attributes

some_attr	Any instance attribute

Alternative Section

some_other_attr	Any other instance attribute

do_something()
Do something

some_attr = None
Any instance attribute

some_other_attr = None
Any other instance attribute

1.2.4 Including a table of contents

Consider for example the following module in `dummy.py`:

```
"""
Some module
-----------

Just a dummy module with some class definition
"""


class MyClass(object):
    """Some class

    With some description"""

    def do_something(self):
        """Do something"""
        pass

    #: Any instance attribute
    some_attr = None
```

(continues on next page)

(continued from previous page)

```
    #: Any other instance attribute
    some_other_attr = None


#: Some module data
large_data = 'Whatever'
```

The *autosummary* flag introduces a small table of contents. So:

```
.. automodule:: dummy
    :members:
    :autosummary:
```

produces *this*. And:

```
.. autoclass:: dummy.SomeClass
    :members:
    :autosummary:
```

produces *this*.

By default, module members are (mainly) grouped according into *Functions*, *Classes* and *Data*, class members are grouped into *Methods* and *Attributes*. But you can also provide alternative section names by connecting to the `autodocsumm-grouper` event. For example, if you include:

```
def example_grouper(app, what, name, obj, section, options, parent):
    import dummy
    if parent is dummy.MyClass and name == 'some_other_attr':
        return 'Alternative Section'


def setup(app):
    app.connect('autodocsumm-grouper', example_grouper)
```

in your *conf.py*, you get *this*.

Note that you can also include the *autosummary* flag in the `autodoc_default_options` configuration value

1.2.5 Including the `__call__` method

Suppose you have a descriptor with a `__call__` method (i.e. somewhat like a method with additional features).

```
"""
Some module
-----------

Just a dummy module with some class definition
"""


class CallableDescriptor(object):
    """A class defining a __call__ method"""

    def __get__(self, instance, owner):
        return self
```

(continues on next page)

(continued from previous page)

```
    def __set__(self, instance, value):
        """Actually not required. We just implement it to ensure the python
        "help" function works well"""
        pass

    def __call__(self, a, b):
        """
        Caller docstring for class attribute

        Parameters
        ----------
        a: any
            dummy parameter
        b: anything else
            second dummy parameter"""
        pass


class MyClass(object):
    """Some class

    With some description"""

    do_something = CallableDescriptor()
```

Then, if you set `autodata_content = 'both'` in your *conf.py* you get via:

```
.. autoclass:: call_demo.MyClass
    :noindex:
    :members:
    :undoc-members:
```

class MyClass

Some class

With some description

Attributes

do_something(a, b)	Caller docstring for class attribute

do_something(*a*, *b*)

Caller docstring for class attribute

Parameters

- **a** (*any*) – dummy parameter
- **b** (*anything else*) – second dummy parameter

1.2.6 Skip large data representations

You can exclude large data representations via the *not_document_data* and *document_data* configuration values.

Suppose you have a dictionary with a very large representation, e.g. in the file *no_data_demo.py*

```
#: very large object
d = dict(zip(range(100), range(100)))
```

which you convert to

d = {0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9, 10: 10, 11: 1
very large object

You can skip this if you specify `not_document_data = ['no_data_demo.d']` in your *conf.py*. Then you get

d
very large object

1.2.7 Generating a summary table without the full documentation

Using one of the `autosummary-...` options (e.g. `autosummary-members`, see *Additional flags for autodoc directives*) let's you create a summary table that points to the documentation in another point of the documentation. You should, however make sure to add the `noindex` flag and to add a `no-members` flag. For our *autodocsumm* module this for example then looks like:

```
.. automodule:: autodocsumm
    :noindex:
    :no-members:
    :autosummary:
    :autosummary-members:
```

which gives us

Classes

AutoSummClassDocumenter(*args)	Class documentor suitable for the *AutoSummDirective*
AutoSummDirective(name, arguments, options, ...)	automodule directive that makes a summary at the beginning of the module
AutoSummModuleDocumenter(*args)	Module documentor suitable for the *AutoSummDirective*
AutosummaryDocumenter()	Abstract class for for extending Documenter methods
CallableAttributeDocumenter(directive, name)	`sphinx.ext.autodoc.AttributeDocumenter` that uses the __call__
CallableDataDocumenter(directive, name[, indent])	`sphinx.ext.autodoc.DataDocumenter` that uses the __call__ attr
NoDataAttributeDocumenter(*args, **kwargs)	AttributeDocumenter that prevents the displaying of large data
NoDataDataDocumenter(*args, **kwargs)	DataDocumenter that prevents the displaying of large data

Functions

dont_document_data(config, fullname)	Check whether the given object should be documented
setup(app)	setup function for using this module as a sphinx extension

Data

member_options	Options of the sphinx.ext.autodoc.ModuleDocumenter that have an

Sphinx extension that defines a new automodule directive with autosummary

The *AutoSummDirective* defined in this extension module allows the same functionality as the automodule and autoclass directives of the sphinx.ext.autodoc module but with an additional *autosummary* option. This option puts a autosummary in the style of the sphinx.ext.autosummary module at the beginning of the class or module. The content of this autosummary is automatically determined by the results of the automodule (or autoclass) directive.

This extension gives also the possibility to choose which data shall be shown and to include the docstring of the '__call__' attribute.

1.2.8 Generating a summary table for the module without nesting

Using the autosummary-no-nesting option, you can generate the autosummary table for a module without generating autosummary tables for members within that module. This is useful when you only want to use the autosummary table as a table of contents for a given page. For the *demo module*, here's an example:

```
.. automodule:: dummy
    :autosummary:
    :members:
    :autosummary-no-nesting:
```

which gives us

Some module

Just a dummy module with some class definition

Classes

MyClass	Some class

Data

large_data	Some module data

class MyClass
Some class

With some description

do_something()
Do something

some_attr = None
Any instance attribute

some_other_attr = None
Any other instance attribute

large_data = 'Whatever'
Some module data

1.3 API Reference

Sphinx extension that defines a new automodule directive with autosummary

The *AutoSummDirective* defined in this extension module allows the same functionality as the automodule and autoclass directives of the `sphinx.ext.autodoc` module but with an additional *autosummary* option. This option puts a autosummary in the style of the `sphinx.ext.autosummary` module at the beginning of the class or module. The content of this autosummary is automatically determined by the results of the automodule (or autoclass) directive.

This extension gives also the possibility to choose which data shall be shown and to include the docstring of the `'__call__'` attribute.

Classes

AutoSummClassDocumenter(*args)	Class documentor suitable for the *AutoSummDirective*
AutoSummDirective(name, arguments, options, ...)	automodule directive that makes a summary at the beginning of the module
AutoSummModuleDocumenter(*args)	Module documentor suitable for the *AutoSummDirective*
AutosummaryDocumenter()	Abstract class for for extending Documenter methods
CallableAttributeDocumenter(directive, name)	`sphinx.ext.autodoc.AttributeDocumenter` that uses the __call__
CallableDataDocumenter(directive, name[, indent])	`sphinx.ext.autodoc.DataDocumenter` that uses the __call__ attr
NoDataAttributeDocumenter(*args, **kwargs)	AttributeDocumenter that prevents the displaying of large data
NoDataDataDocumenter(*args, **kwargs)	DataDocumenter that prevents the displaying of large data

Functions

dont_document_data(config, fullname)	Check whether the given object should be documented
process_signature(obj)	
setup(app)	setup function for using this module as a sphinx extension

Data

member_options	Options of the `sphinx.ext.autodoc.ModuleDocumenter` that have an

class AutoSummClassDocumenter(**args*)

Bases: `sphinx.ext.autodoc.ClassDocumenter`, *autodocsumm.AutosummaryDocumenter*

Class documentor suitable for the *AutoSummDirective*

This class has the same functionality as the base `sphinx.ext.autodoc.ClassDocumenter` class but with an additional *autosummary* option to provide the ability to provide a summary of all methods and attributes at the beginning. It's priority is slightly higher than the one of the ClassDocumenter **Attributes**

member_sections	OrderedDict that includes the autosummary sections
option_spec	original option_spec from sphinx.ext.autodoc.ClassDocumenter
priority	slightly higher priority than

member_sections = {20: 'Classes', 50: 'Methods', 60: 'Attributes'}
OrderedDict that includes the autosummary sections

This dictionary defines the sections for the autosummmary option. The values correspond to the sphinx.ext.autodoc.Documenter.member_order attribute that shall be used for each section.

option_spec = {'autosummary': <function bool_option>, 'autosummary-exclude-members':
original option_spec from sphinx.ext.autodoc.ClassDocumenter but with additional autosummary boolean option

priority = 0.1
slightly higher priority than sphinx.ext.autodoc.ClassDocumenter

class AutoSummDirective(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)
Bases: sphinx.ext.autodoc.directive.AutodocDirective, sphinx.ext.autosummary.Autosummary

automodule directive that makes a summary at the beginning of the module

This directive combines the sphinx.ext.autodoc.directives.AutodocDirective and sphinx.ext.autosummary.Autosummary directives to put a summary of the specified module at the beginning of the module documentation. **Methods**

autosumm_nodes(documenter, ...)	Create the autosummary nodes based on the documenter content
get_items_from_documenters(documenters)	Return the items needed for creating the tables
inject_summ_nodes(doc_nodes, summ_nodes)	Method to inject the autosummary nodes into the autodoc nodes
run()	Run method for the directive

Attributes

autosummary_documenter	Returns the AutosummaryDocumenter subclass that can be used

autosumm_nodes(*documenter*, *grouped_documenters*, *nested*)
Create the autosummary nodes based on the documenter content

Parameters

- **documenter** (*sphinx.ext.autodoc.Documenter*) – The base (module or class) documenter for which to generate the autosummary tables of its members
- **grouped_documenters** (*dict*) – The dictionary as it is returned from the *AutosummaryDocumenter.get_grouped_documenters()* method
- **nested** (*bool*) – If true, autosummary tables will also be created for members.

Returns a mapping from the objects fullname to the corresponding autosummary tables of its members. The objects include the main object of the given *documenter* and the classes that

are defined in it

Return type dict

See also:

AutosummaryDocumenter.get_grouped_documenters(), *inject_summ_nodes()*

autosummary_documenter
Returns the AutosummaryDocumenter subclass that can be used

get_items_from_documenters(*documenters*)
Return the items needed for creating the tables

This method creates the items that are used by the `sphinx.ext.autosummary.Autosummary.get_table()` method by what is taken from the values of the `AutoSummModuleDocumenter.get_grouped_documenters()` method.

Returns A list containing tuples like `(name, signature, summary_string, real_name)` that can be used for the `sphinx.ext.autosummary.Autosummary.get_table()` method.

Return type list

inject_summ_nodes(*doc_nodes*, *summ_nodes*)
Method to inject the autosummary nodes into the autodoc nodes

Parameters

- **doc_nodes** (*list*) – The list of nodes as they are generated by the `sphinx.ext.autodoc.AutodocDirective.run()` method
- **summ_nodes** (*dict*) – The generated autosummary nodes as they are generated by the *autosumm_nodes()* method. Note that *summ_nodes* must only contain the members autosummary tables!

Returns **doc_nodes** – The modified *doc_nodes*

Return type list

Notes

doc_nodes are modified in place and not copied!

run()
Run method for the directive

class AutoSummModuleDocumenter(**args*)
Bases: `sphinx.ext.autodoc.ModuleDocumenter`, *autodocsumm.AutosummaryDocumenter*

Module documentor suitable for the *AutoSummDirective*

This class has the same functionality as the base `sphinx.ext.autodoc.ModuleDocumenter` class but with an additional *autosummary* and the `get_grouped_documenters()` method. It's priority is slightly higher than the one of the ModuleDocumenter. **Attributes**

member_sections	`OrderedDict` that includes the autosummary sections
option_spec	original option_spec from `sphinx.ext.autodoc.ModuleDocumenter`
priority	slightly higher priority than

member_sections = {10: 'Exceptions', 20: 'Classes', 30: 'Functions', 40: 'Data'}
OrderedDict that includes the autosummary sections

This dictionary defines the sections for the autosummmary option. The values correspond to the sphinx.ext.autodoc.Documenter.member_order attribute that shall be used for each section.

option_spec = {'autosummary': <function bool_option>, 'autosummary-exclude-members':
original option_spec from sphinx.ext.autodoc.ModuleDocumenter but with additional autosummary boolean option

priority = 0.1
slightly higher priority than sphinx.ext.autodoc.ModuleDocumenter

class AutosummaryDocumenter
Bases: object

Abstract class for for extending Documenter methods

This classed is used as a base class for Documenters in order to provide the necessary methods for the *AutoSummDirective*. **Attributes**

filter_funcs	List of functions that may filter the members
grouper_funcs	Grouper functions

Methods

get_grouped_documenters([all_members])	Method to return the member documenters

filter_funcs = []
List of functions that may filter the members

get_grouped_documenters(*all_members=False*)
Method to return the member documenters

This method is somewhat like a combination of the sphinx.ext.autodoc.ModuleDocumenter.generate() method and the sphinx.ext.autodoc.ModuleDocumenter.document_members() method. Hence it initializes this instance by importing the object, etc. and it finds the documenters to use for the autosummary option in the same style as the document_members does it.

Returns dictionary whose keys are determined by the member_sections dictionary and whose values are lists of tuples. Each tuple consists of a documenter and a boolean to identify whether a module check should be made describes an attribute or not. The dictionary can be used in the *AutoSummDirective.get_items_from_documenters()* method

Return type dict

Notes

If a sphinx.ext.autodoc.Documenter.member_order value is not in the member_sections dictionary, it will be put into an additional *Miscellaneous* section.

grouper_funcs = []
Grouper functions

class CallableAttributeDocumenter(*directive*, *name*, *indent=u"*)
Bases: sphinx.ext.autodoc.AttributeDocumenter

sphinx.ext.autodoc.AttributeDocumenter that uses the __call__ attr **Methods**

format_args()	Format the argument signature of *self.object*.
get_doc([encoding, ignore])	Reimplemented to include data from the call method

Attributes

priority	float(x) -> floating point number

format_args()
Format the argument signature of *self.object*.

Should return None if the object does not have a signature.

get_doc(*encoding=None*, *ignore=1*)
Reimplemented to include data from the call method

priority = 10.1

class CallableDataDocumenter(*directive*, *name*, *indent=u''*)
Bases: `sphinx.ext.autodoc.DataDocumenter`

`sphinx.ext.autodoc.DataDocumenter` that uses the __call__ attr **Methods**

format_args()	Format the argument signature of *self.object*.
get_doc([encoding, ignore])	Reimplemented to include data from the call method

Attributes

priority	float(x) -> floating point number

format_args()
Format the argument signature of *self.object*.

Should return None if the object does not have a signature.

get_doc(*encoding=None*, *ignore=1*)
Reimplemented to include data from the call method

priority = -9.9

class NoDataAttributeDocumenter(**args*, ***kwargs*)
Bases: *autodocsumm.CallableAttributeDocumenter*

AttributeDocumenter that prevents the displaying of large data **Attributes**

priority	slightly higher priority as the one of the CallableAttributeDocumenter

priority = 10.2
slightly higher priority as the one of the CallableAttributeDocumenter

class NoDataDataDocumenter(**args*, ***kwargs*)
Bases: *autodocsumm.CallableDataDocumenter*

DataDocumenter that prevents the displaying of large data **Attributes**

priority	slightly higher priority as the one of the CallableDataDocumenter

priority = -9.8
slightly higher priority as the one of the CallableDataDocumenter

dont_document_data(*config*, *fullname*)
Check whether the given object should be documented

Parameters

- **config** (*sphinx.Options*) – The configuration
- **fullname** (*str*) – The name of the object

Returns Whether the data of *fullname* should be excluded or not

Return type bool

member_options = set(['exclude-members', 'ignore-module-all', 'imported-members', 'inherite
Options of the `sphinx.ext.autodoc.ModuleDocumenter` that have an effect on the selection of members for the documentation

process_signature(*obj*)

setup(*app*)
setup function for using this module as a sphinx extension

1.4 Installation

Simply install it via `pip`:

```
$ pip install autodocsumm
```

Or you install it via:

```
$ python setup.py install
```

from the source on GitHub.

1.5 Requirements

The package only requires Sphinx to be installed. It has been tested for versions higher than 1.3.

1.6 Quickstart

In order to activate the autodocsumm extension, you have to list it in your `conf.py`:

Listing 1: conf.py

```
extensions = [
    'sphinx.ext.autodoc',
    ...,
    'autodocsumm',
]
```

Once this is done, you can use the `:autosummary:` option for autodoc directives to generate a table at the top, e.g.:

```
.. automodule:: my.awesome.module
    :autosummary:
```

Optionally, you can make autodocsumm active by default for all autodoc directives by adding:

Listing 2: conf.py

```
autodoc_default_options = {
    'autosummary': True,
}
```

1.7 Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

P

R

S