sphinx.ext.autodoc – Include documentation from docstrings

This extension can import the modules you are documenting, and pull in documentation from docstrings in a semi-automatic way.

Note

For Sphinx (actually, the Python interpreter that executes Sphinx) to find your module, it must be importable. That means that the module or the package must be in one of the directories on **`sys.path`** – adapt your **`sys.path`** in the configuration file accordingly.

Warning

`autodoc` **imports** the modules to be documented. If any modules have side effects on import, these will be executed by `autodoc` when `sphinx-build` is run.

If you document scripts (as opposed to library modules), make sure their main routine is protected by a `if __name__ == '__main__'` condition.

For this to work, the docstrings must of course be written in correct reStructuredText. You can then use all of the usual Sphinx markup in the docstrings, and it will end up correctly in the documentation. Together with hand-written documentation, this technique eases the pain of having to maintain two locations for documentation, while at the same time avoiding auto-generated-looking pure API documentation.

If you prefer NumPy or Google style docstrings over reStructuredText, you can also enable the `napoleon` extension. `napoleon` is a preprocessor that converts your docstrings to correct reStructuredText before **autodoc** processes them.

Directives

autodoc provides several directives that are versions of the usual `py:module`, `py:class` and so forth. On parsing time, they import the corresponding module and extract the docstring of the given objects, inserting them into the page source under a suitable `py:module`, `py:class` etc. directive.

Note

Just as `py:class` respects the current `py:module`, `autoclass` will also do so. Likewise, `automethod` will respect the current `py:class`.

.. automodule::
.. autoclass::
.. autoexception::

Document a module, class or exception. All three directives will by default only insert the docstring of the object itself:

```
.. autoclass:: Noodle
```

will produce source like this:

```
.. class:: Noodle

   Noodle's docstring.
```

v: master

The "auto" directives can also contain content of their own, it will be inserted into the resulting non-auto directive source after the docstring (but before any automatic member documentation).

Therefore, you can also mix automatic and non-automatic member documentation, like so:

```
.. autoclass:: Noodle
   :members: eat, slurp

   .. method:: boil(time=10)

      Boil the noodle *time* minutes.
```

Options and advanced usage

- If you want to automatically document members, there's a `members` option:

  ```
  .. automodule:: noodle
     :members:
  ```

  will document all module members (recursively), and

  ```
  .. autoclass:: Noodle
     :members:
  ```

  will document all non-private member functions and properties (that is, those whose name doesn't start with _).

  For modules, `__all__` will be respected when looking for members unless you give the `ignore-module-all` flag option. Without `ignore-module-all`, the order of the members will also be the order in `__all__`.

  You can also give an explicit list of members; only these will then be documented:

  ```
  .. autoclass:: Noodle
     :members: eat, slurp
  ```

- If you want to make the `members` option (or other options described below) the default, see `autodoc_default_options`.

  Tip

  You can use a negated form, `'no-flag'`, as an option of autodoc directive, to disable it temporarily. For example:

  ```
  .. automodule:: foo
     :no-undoc-members:
  ```

- Members without docstrings will be left out, unless you give the `undoc-members` flag option:

  ```
  .. automodule:: noodle
     :members:
     :undoc-members:
  ```

- "Private" members (that is, those named like `_private` or `__private`) will be included if the `private-members` flag option is given.

  New in version 1.1.

- Python "special" members (that is, those named like `__special__`) will be included if the `special-members` flag option is given:

  ```
  .. autoclass:: my.Class
     :members:
     :private-members:
     :special-members:
  ```

v: master

would document both “private” and “special” members of the class.

New in version 1.1.

Changed in version 1.2: The option can now take arguments, i.e. the special members to document.

- For classes and exceptions, members inherited from base classes will be left out when documenting all members, unless you give the `inherited-members` flag option, in addition to `members`:

  ```
  .. autoclass:: Noodle
     :members:
     :inherited-members:
  ```

  This can be combined with `undoc-members` to document *all* available members of the class or module.

  Note: this will lead to markup errors if the inherited members come from a module whose docstrings are not reST formatted.

  New in version 0.3.

- It’s possible to override the signature for explicitly documented callable objects (functions, methods, classes) with the regular syntax that will override the signature gained from introspection:

  ```
  .. autoclass:: Noodle(type)

     .. automethod:: eat(persona)
  ```

  This is useful if the signature from the method is hidden by a decorator.

  New in version 0.4.

- The `automodule`, `autoclass` and `autoexception` directives also support a flag option called `show-inheritance`. When given, a list of base classes will be inserted just below the class signature (when used with `automodule`, this will be inserted for every class that is documented in the module).

  New in version 0.4.

- All autodoc directives support the `noindex` flag option that has the same effect as for standard `py:function` etc. directives: no index entries are generated for the documented object (and all autodocumented members).

  New in version 0.4.

- `automodule` also recognizes the `synopsis`, `platform` and `deprecated` options that the standard `py:module` directive supports.

  New in version 0.5.

- `automodule` and `autoclass` also has an `member-order` option that can be used to override the global value of `autodoc_member_order` for one directive.

  New in version 0.6.

- The directives supporting member documentation also have a `exclude-members` option that can be used to exclude single member names from documentation, if all members are to be documented.

  New in version 0.6.

- In an `automodule` directive with the `members` option set, only module members whose `__module__` attribute is equal to the module name as given to `automodule` will be documented. This is to prevent documentation of imported classes or functions. Set the `imported-members` option if you want to prevent this behavior and document all available members. Note that attributes from imported modules will not be documented, because attribute documentation is discovered by parsing the source file of the current module

  New in version 1.2.

- Add a list of modules in the autodoc_mock_imports to prevent import errors to halt the building process when some external dependencies are not importable at build time.

  New in version 1.3.

.. autofunction::
.. autodecorator::
.. autodata::
.. automethod::
.. autoattribute::

These work exactly like autoclass etc., but do not offer the options used for automatic member documentation.

autodata and autoattribute support the `annotation` option. The option controls how the value of variable is shown. If specified without arguments, only the name of the variable will be printed, and its value is not shown:

```
.. autodata:: CD_DRIVE
   :annotation:
```

If the option specified with arguments, it is printed after the name as a value of the variable:

```
.. autodata:: CD_DRIVE
   :annotation: = your CD device name
```

By default, without `annotation` option, Sphinx tries to obtain the value of the variable and print it after the name.

For module data members and class attributes, documentation can either be put into a comment with special formatting (using a `#:` to start the comment instead of just `#`), or in a docstring *after* the definition. Comments need to be either on a line of their own *before* the definition, or immediately after the assignment *on the same line*. The latter form is restricted to one line only.

This means that in the following class definition, all attributes can be autodocumented:

```
class Foo:
    """Docstring for class Foo."""

    #: Doc comment for class attribute Foo.bar.
    #: It can have multiple lines.
    bar = 1

    flox = 1.5   #: Doc comment for Foo.flox. One line only.

    baz = 2
    """Docstring for class attribute Foo.baz."""

    def __init__(self):
        #: Doc comment for instance attribute qux.
        self.qux = 3

        self.spam = 4
        """Docstring for instance attribute spam."""
```

Changed in version 0.6: autodata and autoattribute can now extract docstrings.

Changed in version 1.1: Comment docs are now allowed on the same line after an assignment.

Changed in version 1.2: autodata and autoattribute have an `annotation` option.

Changed in version 2.0: autodecorator added.

Note

If you document decorated functions or methods, keep in mind that autodoc retrieves its docstrings by import v: master module and inspecting the `__doc__` attribute of the given function or method. That means that if a decorator replaces the decorated function with another, it must copy the original `__doc__` to the new function.

From Python 2.5, **`functools.wraps()`** can be used to create well-behaved decorating functions.

Configuration

There are also config values that you can set:

`autoclass_content`

This value selects what content will be inserted into the main body of an `autoclass` directive. The possible values are:

`"class"`
: Only the class' docstring is inserted. This is the default. You can still document `__init__` as a separate method using `automethod` or the `members` option to `autoclass`.

`"both"`
: Both the class' and the `__init__` method's docstring are concatenated and inserted.

`"init"`
: Only the `__init__` method's docstring is inserted.

New in version 0.3.

If the class has no `__init__` method or if the `__init__` method's docstring is empty, but the class has a `__new__` method's docstring, it is used instead.

New in version 1.4.

`autodoc_member_order`

This value selects if automatically documented members are sorted alphabetical (value `'alphabetical'`), by member type (value `'groupwise'`) or by source order (value `'bysource'`). The default is alphabetical.

Note that for source order, the module must be a Python module with the source code available.

New in version 0.6.

Changed in version 1.0: Support for `'bysource'`.

`autodoc_default_flags`

This value is a list of autodoc directive flags that should be automatically applied to all autodoc directives. The supported flags are `'members'`, `'undoc-members'`, `'private-members'`, `'special-members'`, `'inherited-members'`, `'show-inheritance'`, `'ignore-module-all'` and `'exclude-members'`.

New in version 1.0.

Deprecated since version 1.8: Integrated into `autodoc_default_options`.

`autodoc_default_options`

The default options for autodoc directives. They are applied to all autodoc directives automatically. It must be a dictionary which maps option names to the values. For example:

```
autodoc_default_options = {
    'members': 'var1, var2',
    'member-order': 'bysource',
    'special-members': '__init__',
    'undoc-members': True,
    'exclude-members': '__weakref__'
}
```

Setting `None` or `True` to the value is equivalent to giving only the option name to the directives.

The supported options are `'members'`, `'member-order'`, `'undoc-members'`, `'private-members'`, `'speci` v: master
`'inherited-members'`, `'show-inheritance'`, `'ignore-module-all'`, `'imported-members'` and `'exclude-members'`.

New in version 1.8.

Changed in version 2.0: Accepts `True` as a value.

Changed in version 2.1: Added `'imported-members'`.

`autodoc_docstring_signature`

Functions imported from C modules cannot be introspected, and therefore the signature for such functions cannot be automatically determined. However, it is an often-used convention to put the signature into the first line of the function's docstring.

If this boolean value is set to `True` (which is the default), autodoc will look at the first line of the docstring for functions and methods, and if it looks like a signature, use the line as the signature and remove it from the docstring content.

New in version 1.1.

`autodoc_mock_imports`

This value contains a list of modules to be mocked up. This is useful when some external dependencies are not met at build time and break the building process. You may only specify the root package of the dependencies themselves and omit the sub-modules:

```
autodoc_mock_imports = ["django"]
```

Will mock all imports under the `django` package.

New in version 1.3.

Changed in version 1.6: This config value only requires to declare the top-level modules that should be mocked.

`autodoc_typehints`

This value controls how to represents typehints. The setting takes the following values:

- `'signature'` – Show typehints as its signature (default)
- `'none'` – Do not show typehints

`autodoc_warningiserror`

This value controls the behavior of `sphinx-build -W` during importing modules. If `False` is given, autodoc forcedly suppresses the error if the imported module emits warnings. By default, `True`.

`autodoc_inherit_docstrings`

This value controls the docstrings inheritance. If set to True the docstring for classes or methods, if not explicitly set, is inherited form parents.

The default is `True`.

New in version 1.7.

`suppress_warnings`

autodoc supports to suppress warning messages via `suppress_warnings`. It allows following warnings types in addition:

- autodoc
- autodoc.import_object

Docstring preprocessing

autodoc provides the following additional events:

v: master

`autodoc-process-docstring`(*app*, *what*, *name*, *obj*, *options*, *lines*)

New in version 0.4.

Emitted when autodoc has read and processed a docstring. *lines* is a list of strings – the lines of the processed docstring – that the event handler can modify **in place** to change what Sphinx puts into the output.

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 object itself
- **options** – the options given to the directive: an object with attributes `inherited_members`, `undoc_members`, `show_inheritance` and `noindex` that are true if the flag option of same name was given to the auto directive
- **lines** – the lines of the docstring, see above

`autodoc-process-signature`(*app*, *what*, *name*, *obj*, *options*, *signature*, *return_annotation*)

New in version 0.5.

Emitted when autodoc has formatted a signature for an object. The event handler can return a new tuple `(signature, return_annotation)` to change what Sphinx puts into the output.

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 object itself
- **options** – the options given to the directive: an object with attributes `inherited_members`, `undoc_members`, `show_inheritance` and `noindex` that are true if the flag option of same name was given to the auto directive
- **signature** – function signature, as a string of the form `"(parameter_1, parameter_2)"`, or `None` if introspection didn't succeed and signature wasn't specified in the directive.
- **return_annotation** – function return annotation as a string of the form `" -> annotation"`, or `None` if there is no return annotation

The **`sphinx.ext.autodoc`** module provides factory functions for commonly needed docstring processing in event **`autodoc-process-docstring`**:

`sphinx.ext.autodoc.`**`cut_lines`**(*pre: int*, *post: int = 0*, *what: str = None*) → Callable [source]

Return a listener that removes the first *pre* and last *post* lines of every docstring. If *what* is a sequence of strings, only docstrings of a type in *what* will be processed.

Use like this (e.g. in the `setup()` function of `conf.py`):

```
from sphinx.ext.autodoc import cut_lines
app.connect('autodoc-process-docstring', cut_lines(4, what=['module']))
```

This can (and should) be used in place of **`automodule_skip_lines`**.

`sphinx.ext.autodoc.`**`between`**(*marker: str*, *what: Sequence[str] = None*, *keepempty: bool = False*, *exclude: bool = False*) → Callable [source]

Return a listener that either keeps, or if *exclude* is True excludes, lines between lines that match the *marker* regular expression. If no line matches, the resulting docstring would be empty, so no change will be made unless *keepempty* is true.

If *what* is a sequence of strings, only docstrings of a type in *what* will be processed.

v: master

Skipping members

autodoc allows the user to define a custom method for determining whether a member should be included in the documentation by using the following event:

`autodoc-skip-member`(*app*, *what*, *name*, *obj*, *skip*, *options*)

New in version 0.5.

Emitted when autodoc has to decide whether a member should be included in the documentation. The member is excluded if a handler returns `True`. It is included if the handler returns `False`.

If more than one enabled extension handles the `autodoc-skip-member` event, autodoc will use the first non-`None` value returned by a handler. Handlers should return `None` to fall back to the skipping behavior of autodoc and other enabled extensions.

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 object itself
- **skip** – a boolean indicating if autodoc will skip this member if the user handler does not override the decision
- **options** – the options given to the directive: an object with attributes `inherited_members`, `undoc_members`, `show_inheritance` and `noindex` that are true if the flag option of same name was given to the auto directive