Documentation System

Release 10.6

The Sage Development Team

Apr 01, 2025

CONTENTS

CHAPTER

ONE

SAGE DOC BUILDERS

1.1 Sage docbuild main

This module defines the Sage documentation build command:

```
sage --docbuild [OPTIONS] DOCUMENT (FORMAT | COMMAND)
```

If `FORMAT` is given, it builds `DOCUMENT` in `FORMAT`. If `COMMAND` is given, it returns information about `DOCUMENT`.

Run `sage --docbuild` to get detailed explanations about arguments and options.

Positional arguments:

```
DOCUMENT              name of the document to build. It can be either one of
                      the documents listed by -D or 'file=/path/to/FILE' to
                      build documentation for this specific file.
FORMAT or COMMAND     document output format (or command)
```

Standard options:

```
-h, --help            show a help message and exit
-H, --help-all        show an extended help message and exit
-D, --documents       list all available DOCUMENTs
-F, --formats         list all output FORMATs
-C DOC, --commands DOC
                      list all COMMANDs for DOCUMENT DOC; use 'all' to list all
-i, --inherited       include inherited members in reference manual; may be
                      slow, may fail for PDF output
-u, --underscore      include variables prefixed with '_' in reference
                      manual; may be slow, may fail for PDF output
-j, --mathjax, --jsmath
                      ignored for backwards compatibility
--no-plot             do not include graphics auto-generated using the '.. plot'␣
↪markup
--no-preparsed-examples
                      do not show preparsed versions of EXAMPLES blocks
--include-tests-blocks
                      include TESTS blocks in the reference manual
--no-pdf-links        do not include PDF links in DOCUMENT 'website';
                      FORMATs: html, json, pickle, web
--live-doc            make Sage code blocks live for html FORMAT
--warn-links          issue a warning whenever a link is not properly
```

(continues on next page)

(continued from previous page)

```
                      resolved; equivalent to '--sphinx-opts -n' (sphinx
                      option: nitpicky)
--check-nested        check picklability of nested classes in DOCUMENT 'reference'
--no-prune-empty-dirs
                      do not prune empty directories in the documentation source
--use-cdns            assume internet connection and use CDNs; in particular,
                      use MathJax CDN
-N, --no-colors       do not color output; does not affect children
-q, --quiet           work quietly; same as --verbose=0
-v LEVEL, --verbose LEVEL
                      report progress at LEVEL=0 (quiet), 1 (normal), 2
                      (info), or 3 (debug); does not affect children
-o DIR, --output DIR  if DOCUMENT is a single file ('file=...'), write output
                      to this directory
```

Advanced options:

```
Use these options with care.

-S OPTS, --sphinx-opts OPTS
                      pass comma-separated OPTS to sphinx-build; must precede
                      OPTS with '=', as in '-S=-q,-aE' or '-S="-q,-aE"'
-U, --update-mtimes   before building reference manual, update modification
                      times for auto-generated reST files
-k, --keep-going      Do not abort on errors but continue as much as possible
                      after an error
--all-documents ARG   if ARG is 'reference', list all subdocuments of
                      en/reference. If ARG is 'all', list all main documents
```

class sage_docbuild.__main__.**IntersphinxCache**

Bases: object

Replace sphinx.ext.intersphinx.fetch_inventory by an in-memory cached version.

fetch_inventory(*app*, *uri*, *inv*)

Return the result of `sphinx.ext.intersphinx.fetch_inventory()` from a cache if possible. Otherwise, call `sphinx.ext.intersphinx.fetch_inventory()` and cache the result.

sage_docbuild.__main__.**format_columns**(*lst*, *align='<'*, *cols=None*, *indent=4*, *pad=3*, *width=80*)

Utility function that formats a list as a simple table and returns a Unicode string representation.

The number of columns is computed from the other options, unless it's passed as a keyword argument. For help on Python's string formatter, see

https://docs.python.org/library/string.html#format-string-syntax

sage_docbuild.__main__.**get_formats**()

Return a list of output formats the Sage documentation builder will accept on the command-line.

sage_docbuild.__main__.**help_commands**(*name='all'*)

Append and return a tabular list of commands, if any, the Sage documentation builder can run on the indicated document. The default is to list all commands for all documents.

sage_docbuild.__main__.**help_description**(*s=''*, *compact=False*)

Append and return a brief description of the Sage documentation builder.

If 'compact' is `False`, the function adds a final newline character.

sage_docbuild.__main__.**help_documents**()

Append and return a tabular list of documents, including a shortcut 'all' for all documents, available to the Sage documentation builder.

sage_docbuild.__main__.**help_examples**(*s=''*)

Append and return some usage examples for the Sage documentation builder.

sage_docbuild.__main__.**help_formats**()

Append and return a tabular list of output formats available to the Sage documentation builder.

class sage_docbuild.__main__.**help_message_long**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: `Action`

Print an extended help message for the Sage documentation builder and exits.

class sage_docbuild.__main__.**help_message_short**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: `Action`

Print a help message for the Sage documentation builder.

The message includes command-line usage and a list of options. The message is printed only on the first call. If error is True during this call, the message is printed only if the user hasn't requested a list (e.g., documents, formats, commands).

sage_docbuild.__main__.**help_usage**(*s=''*, *compact=False*)

Append and return a brief usage message for the Sage documentation builder.

If 'compact' is False, the function adds a final newline character.

class sage_docbuild.__main__.**help_wrapper**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: `Action`

A helper wrapper for command-line options to the Sage documentation builder that print lists, such as document names, formats, and document-specific commands.

sage_docbuild.__main__.**main**()

sage_docbuild.__main__.**setup_logger**(*verbose=1*, *color=True*)

Set up a Python Logger instance for the Sage documentation builder.

The optional argument sets logger's level and message format.

EXAMPLES:

```
sage: from sage_docbuild.__main__ import setup_logger, logger
sage: setup_logger()
sage: type(logger)
<class 'logging.Logger'>
```

```
>>> from sage.all import *
>>> from sage_docbuild.__main__ import setup_logger, logger
>>> setup_logger()
>>> type(logger)
<class 'logging.Logger'>
```

sage_docbuild.__main__.**setup_parser**()

Set up and return a command-line ArgumentParser instance for the Sage documentation builder.

1.2 Documentation builders

This module is the starting point for building documentation, and is responsible to figure out what to build and with which options. The actual documentation build for each individual document is then done in a subprocess call to Sphinx, see *builder_helper()*. Note that

- The builders are configured with `build_options.py`;
- The Sphinx subprocesses are configured in `conf.py`.

DocBuilder is the base class of all Builders. It has builder helpers `html()`, `latex()`, `pdf()`, `inventory()`, etc, which are invoked depending on the output type. Each type corresponds with the Sphinx builder format, except that `pdf` is Sphinx latex builder plus compiling latex to pdf. Note that Sphinx inventory builder is not native to Sphinx but provided by Sage. See `sage_docbuild/ext/inventory_builder.py`. The Sphinx inventory builder is a dummy builder with no actual output but produces doctree files in `local/share/doctree` and `inventory.inv` inventory files in `local/share/inventory`.

The reference manual is built in two passes, first by *ReferenceBuilder* with `inventory` output type and secondly with``html`` output type. The *ReferenceBuilder* itself uses *ReferenceTopBuilder* and *ReferenceSubBuilder* to build subcomponents of the reference manual. The *ReferenceSubBuilder* examines the modules included in the subcomponent by comparing the modification times of the module files with the times saved in `local/share/doctree/reference.pickle` from the previous build. Then new rst files are generated for new and updated modules. See `get_new_and_updated_modules()`.

After Issue #31948, when Sage is built, *ReferenceBuilder* is not used and its responsibility is now taken by the `Makefile` in `$SAGE_ROOT/src/doc`.

class sage_docbuild.builders.**AllBuilder**

Bases: `object`

A class used to build all of the documentation.

get_all_documents()

Return a list of all of the documents.

A document is a directory within one of the language subdirectories of SAGE_DOC_SRC specified by the global LANGUAGES variable.

EXAMPLES:

```
sage: from sage_docbuild.builders import AllBuilder
sage: documents = AllBuilder().get_all_documents()
sage: 'en/tutorial' in documents  # optional - sage_spkg
True
sage: documents[0] == 'en/reference'
True
```

```
>>> from sage.all import *
>>> from sage_docbuild.builders import AllBuilder
>>> documents = AllBuilder().get_all_documents()
>>> 'en/tutorial' in documents  # optional - sage_spkg
True
>>> documents[Integer(0)] == 'en/reference'
True
```

class sage_docbuild.builders.**DocBuilder**(*name*, *lang='en'*)

Bases: object

INPUT:

- name – the name of a subdirectory in SAGE_DOC_SRC, such as 'tutorial' or 'bordeaux_2008'
- lang – (default "en") the language of the document.

changes(**args*, ***kwds*)

clean(**args*)

html(**args*, ***kwds*)

htmlhelp(**args*, ***kwds*)

inventory(**args*, ***kwds*)

json(**args*, ***kwds*)

latex(**args*, ***kwds*)

linkcheck(**args*, ***kwds*)

pdf()

Build the PDF files for this document.

This is done by first (re)-building the LaTeX output, going into that LaTeX directory, and running 'make all-pdf' there.

EXAMPLES:

```
sage: from sage_docbuild.builders import DocBuilder
sage: b = DocBuilder('tutorial')
sage: b.pdf() #not tested
```

```
>>> from sage.all import *
>>> from sage_docbuild.builders import DocBuilder
>>> b = DocBuilder('tutorial')
>>> b.pdf() #not tested
```

pickle(**args*, ***kwds*)

web(**args*, ***kwds*)

class sage_docbuild.builders.**ReferenceBuilder**(*name*, *lang='en'*)

Bases: *AllBuilder*

This class builds the reference manual. It uses DocBuilder to build the top-level page and ReferenceSubBuilder for each sub-component.

get_all_documents(*refdir*)

Return a list of all reference manual components to build.

We add a component name if it's a subdirectory of the manual's directory and contains a file named 'index.rst'.

We return the largest component (most subdirectory entries) first since they will take the longest to build.

EXAMPLES:

```
sage: from sage_docbuild.builders import ReferenceBuilder
sage: b = ReferenceBuilder('reference')
sage: refdir = os.path.join(os.environ['SAGE_DOC_SRC'], 'en', b.name)  #␣
↪optional - sage_spkg
sage: sorted(b.get_all_documents(refdir))  # optional - sage_spkg
['reference/algebras',
 'reference/arithgroup',
 ...,
 'reference/valuations']
```

```
>>> from sage.all import *
>>> from sage_docbuild.builders import ReferenceBuilder
>>> b = ReferenceBuilder('reference')
>>> refdir = os.path.join(os.environ['SAGE_DOC_SRC'], 'en', b.name)  #␣
↪optional - sage_spkg
>>> sorted(b.get_all_documents(refdir))  # optional - sage_spkg
['reference/algebras',
 'reference/arithgroup',
 ...,
 'reference/valuations']
```

class sage_docbuild.builders.**ReferenceSubBuilder**(**args*, ***kwds*)

Bases: *DocBuilder*

This class builds sub-components of the reference manual. It is responsible for making sure that the auto generated reST files for the Sage library are up to date.

When building any output, we must first go through and check to see if we need to update any of the autogenerated reST files. There are two cases where this would happen:

1. A new module gets added to one of the toctrees.
2. The actual module gets updated and possibly contains a new title.

auto_rest_filename(*module_name*)

Return the name of the file associated to a given module

EXAMPLES:

```
sage: from sage_docbuild.builders import ReferenceSubBuilder
sage: ReferenceSubBuilder("reference").auto_rest_filename("sage.combinat.
↪partition")
'.../en/reference/sage/combinat/partition.rst'
```

```
>>> from sage.all import *
>>> from sage_docbuild.builders import ReferenceSubBuilder
>>> ReferenceSubBuilder("reference").auto_rest_filename("sage.combinat.
```

(continues on next page)

(continued from previous page)

```
↪partition")
'.../en/reference/sage/combinat/partition.rst'
```

cache_filename()

Return the filename where the pickle of the reference cache is stored.

clean_auto()

Remove all autogenerated reST files.

get_all_included_modules()

Return an iterator for all modules which are included in the reference manual.

get_all_rst_files(*exclude_sage=True*)

Return an iterator for all rst files which are not autogenerated.

get_cache()

Retrieve the reference cache which contains the options previously used by the reference builder.

If it doesn't exist, then we just return an empty dictionary. If it is corrupted, return an empty dictionary.

get_modified_modules()

Return an iterator for all the modules that have been modified since the documentation was last built.

get_module_docstring_title(*module_name*)

Return the title of the module from its docstring.

get_modules(*filename*)

Given a filename for a reST file, return an iterator for all of the autogenerated reST files that it includes.

get_new_and_updated_modules()

Return an iterator for all new and updated modules that appear in the toctrees, and remove obsolete old modules.

get_sphinx_environment()

Return the Sphinx environment for this project.

get_unincluded_modules()

Return an iterator for all the modules in the Sage library which are not included in the reference manual.

print_included_modules()

Print all of the modules that are included in the Sage reference manual.

print_modified_modules()

Print a list of all the modules that have been modified since the documentation was last built.

print_new_and_updated_modules()

Print all the modules that appear in the toctrees that are newly included or updated.

print_unincluded_modules()

Print all of the modules which are not included in the Sage reference manual.

save_cache()

Pickle the current reference cache for later retrieval.

update_mtimes()

Update the modification times for reST files in the Sphinx environment for this project.

write_auto_rest_file(*module_name*)

Write the autogenerated reST file for module_name.

class sage_docbuild.builders.**ReferenceTopBuilder**(**args*, ***kwds*)

Bases: *DocBuilder*

This class builds the top-level page of the reference manual.

html()

Build the top-level document.

class sage_docbuild.builders.**SingleFileBuilder**(*path*)

Bases: *DocBuilder*

This is the class used to build the documentation for a single user-specified file. If the file is called 'foo.py', then the documentation is built in `DIR/foo/` if the user passes the command line option "-o DIR", or in `DOT_SAGE/docbuild/foo/` otherwise.

class sage_docbuild.builders.**WebsiteBuilder**(*name*, *lang='en'*)

Bases: *DocBuilder*

clean()

When we clean the output for the website index, we need to remove all of the HTML that were placed in the parent directory.

In addition, remove the index file installed into the root doc directory.

html()

After we have finished building the website index page, we copy everything one directory up, that is, to the base diectory `html/en`.

In addition, an index file is installed into the root doc directory.

Thus we have three index.html files:

html/en/website/index.html (not used) html/en/index.html (base directory) index.html (root doc directory)

pdf()

Build the website hosting pdf docs.

sage_docbuild.builders.**build_many**(*target*, *args*, *processes=None*)

Thin wrapper around $sage_docbuild.utils.build_many$ which uses the docbuild settings `NUM_THREADS` and `ABORT_ON_ERROR`.

sage_docbuild.builders.**build_other_doc**(*args*)

sage_docbuild.builders.**build_ref_doc**(*args*)

sage_docbuild.builders.**builder_helper**(*type*)

Return a function which builds the documentation for output type `type`.

sage_docbuild.builders.**get_builder**(*name*)

Return an appropriate *Builder* object for the document `name`.

DocBuilder and its subclasses do all the real work in building the documentation.

sage_docbuild.builders.**get_documents**()

Return a list of document names the Sage documentation builder will accept as command-line arguments.

1.3 Build options

This module defines options for building Sage documentation.

1.4 Sphinx build script

This is Sage's version of the `sphinx-build` script. We redirect `stdout` and `stderr` to our own logger, and remove some unwanted chatter.

class sage_docbuild.sphinxbuild.**SageSphinxLogger**(*stream*, *prefix*)

Bases: `object`

This implements the file object interface to serve as `sys.stdout`/`sys.stderr` replacement.

ansi_escape_sequence = re.compile('\n \\x1b # ESC\n \\[# CSI sequence starts\n [0-?]* # parameter bytes\n [-/]* # intermediate bytes\n [@-~] # final byte\n ', re.VERBOSE)

ansi_escape_sequence_color = re.compile('\n \\x1b # ESC\n \\[# CSI sequence starts\n [0-9;]* # parameter bytes\n # intermediate bytes\n m # final byte\n ', re.VERBOSE)

close()

closed = False

encoding = None

flush()

isatty()

mode = 'w'

name = '<log>'

newlines = None

prefix_len = 9

raise_errors()

Raise an exceptions if any errors have been found while parsing the Sphinx output.

EXAMPLES:

```
sage: from sys import stdout
sage: from sage_docbuild.sphinxbuild import SageSphinxLogger
sage: logger = SageSphinxLogger(stdout, "doctesting")
sage: logger._log_line("This is a SEVERE error\n")
[doctestin] This is a SEVERE error
sage: logger.raise_errors()
Traceback (most recent call last):
...
OSError: This is a SEVERE error
```

```
>>> from sage.all import *
>>> from sys import stdout
>>> from sage_docbuild.sphinxbuild import SageSphinxLogger
>>> logger = SageSphinxLogger(stdout, "doctesting")
>>> logger._log_line("This is a SEVERE error\n")
[doctestin] This is a SEVERE error
>>> logger.raise_errors()
Traceback (most recent call last):
...
OSError: This is a SEVERE error
```

softspace = 0

write(*str*)

writelines(*sequence*)

sage_docbuild.sphinxbuild.**runsphinx**()

sage_docbuild.sphinxbuild.**term_width_line**(*text*)

1.5 Sphinx build configuration

This file contains configuration needed to customize Sphinx input and output behavior.

class sage_docbuild.conf.**Ignore**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

Bases: SphinxDirective

has_content = True

May the directive have content?

run()

class sage_docbuild.conf.**SagecodeTransform**(*document*, *startnode=None*)

Bases: SphinxTransform

Transform a code block to a live code block enabled by jupyter-sphinx.

Effectively a code block like:

```
EXAMPLE::

    sage: 1 + 1
    2
```

is transformed into:

```
EXAMPLE::

    sage: 1 + 1
    2

.. ONLY:: html
```

(continues on next page)

(continued from previous page)

```
.. JUPYTER-EXECUTE::
    :hide-code:
    :hide-output:
    :raises:
    :stderr:

    1 + 1
```

enabling live execution of the code.

apply()

default_priority = 170

Numerical priority of this transform, 0 through 999 (override).

sage_docbuild.conf.**add_page_context**(*app*, *pagename*, *templatename*, *context*, *doctree*)

sage_docbuild.conf.**call_intersphinx**(*app*, *env*, *node*, *contnode*)

Call intersphinx and make links between Sage manuals relative.

sage_docbuild.conf.**check_nested_class_picklability**(*app*, *what*, *name*, *obj*, *skip*, *options*)

Print a warning if pickling is broken for nested classes.

sage_docbuild.conf.**debug_inf**(*app*, *message*)

sage_docbuild.conf.**feature_tags**()

sage_docbuild.conf.**find_sage_dangling_links**(*app*, *env*, *node*, *contnode*)

Try to find dangling link in local module imports or all.py.

sage_docbuild.conf.**linkcode_resolve**(*domain*, *info*)

sage_docbuild.conf.**set_intersphinx_mappings**(*app*, *config*)

Add precompiled inventory (the objects.inv)

sage_docbuild.conf.**setup**(*app*)

sage_docbuild.conf.**skip_member**(*app*, *what*, *name*, *obj*, *skip*, *options*)

To suppress Sphinx warnings / errors, we

- Don't include [aliases of] builtins.
- Don't include the docstring for any nested class which has been inserted into its module by `sage.misc.NestedClassMetaclass` only for pickling. The class will be properly documented inside its surrounding class.
- Optionally, check whether pickling is broken for nested classes.
- Optionally, include objects whose name begins with an underscore ('_'), i.e., "private" or "hidden" attributes, methods, etc.

Otherwise, we abide by Sphinx's decision. Note: The object `obj` is excluded (included) if this handler returns True (False).

1.6 Utilities

exception sage_docbuild.utils.**RemoteException**(*tb: str*)

Bases: Exception

Raised if an exception occurred in one of the child processes.

tb: str

class sage_docbuild.utils.**RemoteExceptionWrapper**(*exc: BaseException*)

Bases: object

Used by child processes to capture exceptions thrown during execution and report them to the main process, including the correct traceback.

exc: BaseException

tb: str

exception sage_docbuild.utils.**WorkerDiedException**(*message: str | None*, *original_exception: BaseException | None = None*)

Bases: RuntimeError

Raised if a worker process dies unexpected.

original_exception: BaseException | None

sage_docbuild.utils.**build_many**(*target*, *args*, *processes=None*)

Map a list of arguments in `args` to a single-argument target function `target` in parallel using `multiprocessing.cpu_count()` (or `processes` if given) simultaneous processes.

This is a simplified version of `multiprocessing.Pool.map` from the Python standard library which avoids a couple of its pitfalls. In particular, it can abort (with a `RuntimeError`) without hanging if one of the worker processes unexpectedly dies. It also has semantics equivalent to `maxtasksperchild=1`; that is, one process is started per argument. As such, this is inefficient for processing large numbers of fast tasks, but appropriate for running longer tasks (such as doc builds) which may also require significant cleanup.

It also avoids starting new processes from a pthread, which results in at least one known issue:

- When PARI is built with multi-threading support, forking a Sage process from a thread leaves the main Pari interface instance broken (see Issue #26608#comment:38).

In the future this may be replaced by a generalized version of the more robust parallel processing implementation from `sage.doctest.forker`.

EXAMPLES:

```
sage: from sage_docbuild.utils import build_many
sage: def target(N):
....:     import time
....:     time.sleep(float(0.1))
....:     print('Processed task %s' % N)
sage: _ = build_many(target, range(8), processes=8)
Processed task ...
Processed task ...
Processed task ...
Processed task ...
Processed task ...
Processed task ...
```

(continues on next page)

(continued from previous page)

```
Processed task ...
Processed task ...
```

```
>>> from sage.all import *
>>> from sage_docbuild.utils import build_many
>>> def target(N):
...     import time
...     time.sleep(float(RealNumber('0.1')))
...     print('Processed task %s' % N)
>>> _ = build_many(target, range(Integer(8)), processes=Integer(8))
Processed task ...
Processed task ...
Processed task ...
Processed task ...
Processed task ...
Processed task ...
Processed task ...
Processed task ...
```

This version can also return a result, and thus can be used as a replacement for `multiprocessing.Pool.map` (i.e. it still blocks until the result is ready):

```
sage: def square(N):
....:     return N * N
sage: build_many(square, range(100))
[0, 1, 4, 9, ..., 9604, 9801]
```

```
>>> from sage.all import *
>>> def square(N):
...     return N * N
>>> build_many(square, range(Integer(100)))
[0, 1, 4, 9, ..., 9604, 9801]
```

If the target function raises an exception in any of the workers, `build_many` raises that exception and all other results are discarded. Any in-progress tasks may still be allowed to complete gracefully before the exception is raised:

```
sage: def target(N):
....:     import time, os, signal
....:     if N == 4:
....:         # Task 4 is a poison pill
....:         1 / 0
....:     else:
....:         time.sleep(float(0.5))
....:         print('Processed task %s' % N)
```

```
>>> from sage.all import *
>>> def target(N):
...     import time, os, signal
...     if N == Integer(4):
...         # Task 4 is a poison pill
...         Integer(1) / Integer(0)
```

(continues on next page)

(continued from previous page)

```
...     else:
...         time.sleep(float(RealNumber('0.5')))
...         print('Processed task %s' % N)
```

Note: In practice this test might still show output from the other worker processes before the poison-pill is executed. It may also display the traceback from the failing process on stderr. However, due to how the doctest runner works, the doctest will only expect the final exception:

```
sage: build_many(target, range(8), processes=8)
Traceback (most recent call last):
...
    raise ZeroDivisionError("rational division by zero")
 ZeroDivisionError: rational division by zero
...
    raise worker_exc.original_exception
ZeroDivisionError: rational division by zero
```

```
>>> from sage.all import *
>>> build_many(target, range(Integer(8)), processes=Integer(8))
Traceback (most recent call last):
...
    raise ZeroDivisionError("rational division by zero")
 ZeroDivisionError: rational division by zero
...
    raise worker_exc.original_exception
ZeroDivisionError: rational division by zero
```

Similarly, if one of the worker processes dies unexpectedly otherwise exits non-zero (e.g. killed by a signal) any in-progress tasks will be completed gracefully, but then a `RuntimeError` is raised and pending tasks are not started:

```
sage: def target(N):
....:     import time, os, signal
....:     if N == 4:
....:         # Task 4 is a poison pill
....:         os.kill(os.getpid(), signal.SIGKILL)
....:     else:
....:         time.sleep(float(0.5))
....:         print('Processed task %s' % N)
sage: build_many(target, range(8), processes=8)
Traceback (most recent call last):
...
WorkerDiedException: worker for 4 died with non-zero exit code -9
```

```
>>> from sage.all import *
>>> def target(N):
...     import time, os, signal
...     if N == Integer(4):
...         # Task 4 is a poison pill
...         os.kill(os.getpid(), signal.SIGKILL)
...     else:
...         time.sleep(float(RealNumber('0.5')))
```

(continues on next page)

(continued from previous page)

```
...         print('Processed task %s' % N)
>>> build_many(target, range(Integer(8)), processes=Integer(8))
Traceback (most recent call last):
...
WorkerDiedException: worker for 4 died with non-zero exit code -9
```

CHAPTER

TWO

SPHINX EXTENSIONS

2.1 Inventory builder

A customized builder which only generates intersphinx "object.inv" inventory files. The documentation files are not written.

class sage_docbuild.ext.inventory_builder.**InventoryBuilder**(*app: Sphinx*, *env: BuildEnvironment*)

Bases: DummyBuilder

A customized builder which only generates intersphinx "object.inv" inventory files. The documentation files are not written.

epilog: str = 'The inventory file is in %(outdir)s.'

The message emitted upon successful build completion. This can be a printf-style template string with the following keys: outdir, project

finish()

Only write the inventory files.

format: str = 'inventory'

The builder's output format, or '' if no document output is produced. This is commonly the file extension, e.g. "html", though any string value is accepted. The builder's format string can be used by various components such as SphinxPostTransform or extensions to determine their compatibility with the builder.

get_outdated_docs()

Return an iterable of output files that are outdated.

get_target_uri(*docname*, *typ=None*)

Return the target URI for a document name.

name: str = 'inventory'

The builder's name. This is the value used to select builders on the command line.

sage_docbuild.ext.inventory_builder.**setup**(*app*)

2.2 Sage autodoc extension

This is sphinx.ext.autodoc extension modified for Sage objects.

The original header of sphinx.ext.autodoc:

Extension to create automatic documentation from code docstrings.

Automatically insert docstrings for functions, classes or whole modules into the doctree, thus avoiding duplication between docstrings and documentation for those who like elaborate docstrings.

This module is currently based on `sphinx.ext.autodoc` from Sphinx version 8.0.2. Compare (do diff) with the upstream source file sphinx/ext/autodoc/__init__.py.

In the source file of this module, major modifications are delimited by a pair of comment dividers. To lessen maintenance burden, we aim at reducing the modifications.

AUTHORS:

- ? (before 2011): initial version derived from sphinx.ext.autodoc
- Johan S. R. Nielsen: support for _sage_argspec_
- Simon King (2011-04): use sageinspect; include public cython attributes only in the documentation if they have a docstring
- Kwankyu Lee (2018-12-26, 2022-11-08): rebased on the latest sphinx.ext.autodoc
- Kwankyu Lee (2024-02-14): rebased on Sphinx 7.2.6
- François Bissey (2024-08-24): rebased on Sphinx 8.0.2
- François Bissey (2024-09-10): Tweaks to support python 3.9 (and older sphinx) as well
- François Bissey (2024-11-12): rebased on Sphinx 8.1.3 (while trying to keep python 3.9 compatibility)
- François Bissey (2025-02-24): Remove python 3.9 support hacks, making us closer to upstream

class sage_docbuild.ext.sage_autodoc.**AttributeDocumenter**(*directive: DocumenterBridge*, *name: str*, *indent: str = ''*)

Bases: *GenericAliasMixin*, *SlotsMixin*, *RuntimeInstanceAttributeMixin*, *UninitializedInstanceAttributeMixin*, *NonDataDescriptorMixin*, *DocstringStripSignatureMixin*, *ClassLevelDocumenter*

Specialized Documenter subclass for attributes.

add_content(*more_content*)

add_directive_header(*sig*)

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

document_members(*all_members=False*)

get_attribute_comment(*parent*, *attrname*)

get_doc()

get_real_modname()

import_object(*raiseerror=False*)

static is_function_or_method(*obj*)

member_order = 60

order if autodoc_member_order is set to 'groupwise'

objtype = 'attribute'

name by which the directive is called (auto…) and the default generated directive name

option_spec: ClassVar[OptionSpec] = {'annotation': <function annotation_option>, 'no-index': <function bool_option>, 'no-value': <function bool_option>, 'noindex': <function bool_option>}

priority = 10

priority if multiple documenters return True from can_document_member

should_suppress_value_header()

update_annotations(*parent*)

Update __annotations__ to support type_comment and so on.

class sage_docbuild.ext.sage_autodoc.**ClassDocumenter**(**args: Any*)

Bases: *DocstringSignatureMixin*, *ModuleLevelDocumenter*

Specialized Documenter subclass for classes.

add_content(*more_content*)

add_directive_header(*sig*)

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

document_members(*all_members=False*)

format_args(***kwargs*)

format_signature(***kwargs*)

generate(*more_content=None*, *real_modname=None*, *check_module=False*, *all_members=False*)

get_canonical_fullname()

get_doc()

get_object_members(*want_all*)

get_overloaded_signatures()

get_variable_comment()

import_object(*raiseerror=False*)

member_order = 20

order if autodoc_member_order is set to 'groupwise'

objtype = 'class'

name by which the directive is called (auto…) and the default generated directive name

option_spec: ClassVar[OptionSpec] = {'class-doc-from': <function class_doc_from_option>, 'exclude-members': <function exclude_members_option>, 'inherited-members': <function inherited_members_option>, 'member-order': <function member_order_option>, 'members': <function members_option>, 'no-index': <function bool_option>, 'noindex': <function bool_option>, 'private-members': <function members_option>, 'show-inheritance': <function bool_option>, 'special-members': <function members_option>, 'undoc-members': <function bool_option>}

priority = 15

priority if multiple documenters return True from can_document_member

class sage_docbuild.ext.sage_autodoc.**ClassLevelDocumenter**(*directive: DocumenterBridge*, *name: str*, *indent: str = ''*)

Bases: *Documenter*

Specialized Documenter subclass for objects on class level (methods, attributes).

resolve_name(*modname*, *parents*, *path*, *base*)

class sage_docbuild.ext.sage_autodoc.**DataDocumenter**(*directive: DocumenterBridge*, *name: str*, *indent: str = ''*)

Bases: *GenericAliasMixin*, *UninitializedGlobalVariableMixin*, *ModuleLevelDocumenter*

Specialized Documenter subclass for data items.

add_content(*more_content*)

add_directive_header(*sig*)

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

document_members(*all_members=False*)

get_doc()

get_module_comment(*attrname*)

get_real_modname()

import_object(*raiseerror=False*)

member_order = 40

order if autodoc_member_order is set to 'groupwise'

objtype = 'data'

name by which the directive is called (auto…) and the default generated directive name

option_spec: ClassVar[OptionSpec] = {'annotation': <function annotation_option>, 'no-index': <function bool_option>, 'no-value': <function bool_option>, 'noindex': <function bool_option>}

priority = -10

priority if multiple documenters return True from can_document_member

should_suppress_value_header()

update_annotations(*parent*)

Update __annotations__ to support type_comment and so on.

class sage_docbuild.ext.sage_autodoc.**DataDocumenterMixinBase**

Bases: *object*

config: Config

env: BuildEnvironment

modname: str

object: Any

objpath: list[str]

parent: Any

should_suppress_directive_header()

Check directive header should be suppressed.

should_suppress_value_header()

Check :value: header should be suppressed.

update_content(*more_content*)

Update docstring, for example with TypeVar variance.

class sage_docbuild.ext.sage_autodoc.**DecoratorDocumenter**(*directive: DocumenterBridge*, *name: str*, *indent: str = ''*)

Bases: *FunctionDocumenter*

Specialized Documenter subclass for decorator functions.

format_args(***kwargs*)

objtype = 'decorator'

name by which the directive is called (auto…) and the default generated directive name

priority = -1

priority if multiple documenters return True from can_document_member

class sage_docbuild.ext.sage_autodoc.**DocstringSignatureMixin**

Bases: object

Mixin for FunctionDocumenter and MethodDocumenter to provide the feature of reading the signature from the docstring.

format_signature(***kwargs*)

get_doc()

class sage_docbuild.ext.sage_autodoc.**DocstringStripSignatureMixin**

Bases: *DocstringSignatureMixin*

Mixin for AttributeDocumenter to provide the feature of stripping any function signature from the docstring.

format_signature(***kwargs*)

class sage_docbuild.ext.sage_autodoc.**Documenter**(*directive: DocumenterBridge*, *name: str*, *indent: str = ''*)

Bases: object

A Documenter knows how to autodocument a single object type. When registered with the AutoDirective, it will be used to document objects of that type when needed by autodoc.

Its *objtype* attribute selects what auto directive it is assigned to (the directive name is 'auto' + objtype), and what directive it generates by default, though that can be overridden by an attribute called *directivetype*.

A Documenter has an *option_spec* that works like a docutils directive's; in fact, it will be used to parse an auto directive's options that matches the Documenter.

add_content(*more_content*)

Add content from docstrings, attribute documentation and user.

add_directive_header(*sig*)

Add the directive header and options to the generated content.

add_line(*line*, *source*, **lineno*)

Append one line of generated reST to the output.

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

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

check_module()

Check if *self.object* is really defined in the module given by *self.modname*.

content_indent = ' '

indentation by which to indent the directive content

document_members(*all_members=False*)

Generate reST for member documentation.

If *all_members* is True, document all members, else those given by *self.options.members*.

property documenters: dict[str, type[*Documenter*]]

Returns registered Documenter classes

filter_members(*members*, *want_all*)

Filter the given member list.

Members are skipped if

- they are private (except if given explicitly or the private-members option is set)
- they are special methods (except if given explicitly or the special-members option is set)
- they are undocumented (except if the undoc-members option is set)

The user can override the skipping decision by connecting to the `autodoc-skip-member` event.

format_args(***kwargs*)

Format the argument signature of *self.object*.

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

format_name()

Format the name of *self.object*.

This normally should be something that can be parsed by the generated directive, but doesn't need to be (Sphinx will display it unparsed then).

format_signature(***kwargs*)

Format the signature (arguments and return annotation) of the object.

Let the user process it via the `autodoc-process-signature` event.

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.

If *more_content* is given, include that content. If *real_modname* is given, use that module name to find attribute docs. If *check_module* is True, only generate if the object is defined in the module name it is imported from. If *all_members* is True, document all members.

get_attr(*obj*, *name*, **defargs*)

getattr() override for types such as Zope interfaces.

get_doc()

Decode and return lines of the docstring(s) for the object.

When it returns None, autodoc-process-docstring will not be called for this object.

get_object_members(*want_all*)

Return $(members_check_module, members)$ where $members$ is a list of $(membername, member)$ pairs of the members of *self.object*.

If *want_all* is True, return all members. Else, only return those members given by *self.options.members* (which may also be None).

get_real_modname()

Get the real module name of an object to document.

It can differ from the name of the module through which the object was imported.

get_sourcename()

import_object(*raiseerror=False*)

Import the object given by *self.modname* and *self.objpath* and set it as *self.object*.

Returns True if successful, False if an error occurred.

member_order = 0

order if autodoc_member_order is set to 'groupwise'

objtype = 'object'

name by which the directive is called (auto…) and the default generated directive name

option_spec: ClassVar[OptionSpec] = {'no-index': <function bool_option>, 'noindex': <function bool_option>}

parse_name()

Determine what module to import and what attribute to document.

Returns True and sets *self.modname*, *self.objpath*, *self.fullname*, *self.args* and *self.retann* if parsing and resolving was successful.

priority = 0

priority if multiple documenters return True from can_document_member

process_doc(*docstrings*)

Let the user process the docstrings before adding them.

resolve_name(*modname*, *parents*, *path*, *base*)

Resolve the module and name of the object to document given by the arguments and the current module/class.

Must return a pair of the module name and a chain of attributes; for example, it would return `('zipfile', ['ZipFile', 'open'])` for the `zipfile.ZipFile.open` method.

sort_members(*documenters*, *order*)

Sort the given member list.

titles_allowed = True

true if the generated content may contain titles

class sage_docbuild.ext.sage_autodoc.**ExceptionDocumenter**(**args: Any*)

Bases: *ClassDocumenter*

Specialized ClassDocumenter subclass for exceptions.

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

member_order = 10

order if autodoc_member_order is set to 'groupwise'

objtype = 'exception'

name by which the directive is called (auto…) and the default generated directive name

priority = 20

priority if multiple documenters return True from can_document_member

class sage_docbuild.ext.sage_autodoc.**FunctionDocumenter**(*directive: DocumenterBridge*, *name: str*, *indent: str = ''*)

Bases: *DocstringSignatureMixin*, *ModuleLevelDocumenter*

Specialized Documenter subclass for functions.

add_directive_header(*sig*)

annotate_to_first_argument(*func*, *typ*)

Annotate type hint to the first argument of function if needed.

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

document_members(*all_members=False*)

format_args(***kwargs*)

format_signature(***kwargs*)

member_order = 30

order if autodoc_member_order is set to 'groupwise'

merge_default_value(*actual*, *overload*)

Merge default values of actual implementation to the overload variants.

objtype = 'function'

name by which the directive is called (auto…) and the default generated directive name

class sage_docbuild.ext.sage_autodoc.**GenericAliasMixin**

Bases: *DataDocumenterMixinBase*

Mixin for DataDocumenter and AttributeDocumenter to provide the feature for supporting GenericAliases.

should_suppress_directive_header()

update_content(*more_content*)

sage_docbuild.ext.sage_autodoc.**INSTANCEATTR = <object object>**

The base class of the class hierarchy.

When called, it accepts no arguments and returns a new featureless instance that has no instance attributes and cannot be given any.

class sage_docbuild.ext.sage_autodoc.**MethodDocumenter**(*directive: DocumenterBridge*, *name: str*, *indent: str = ''*)

Bases: *DocstringSignatureMixin*, *ClassLevelDocumenter*

Specialized Documenter subclass for methods (normal, static and class).

add_directive_header(*sig*)

annotate_to_first_argument(*func*, *typ*)

Annotate type hint to the first argument of function if needed.

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

directivetype = 'method'

document_members(*all_members=False*)

format_args(***kwargs*)

get_doc()

import_object(*raiseerror=False*)

member_order = 50

order if autodoc_member_order is set to 'groupwise'

merge_default_value(*actual*, *overload*)

Merge default values of actual implementation to the overload variants.

objtype = 'method'

name by which the directive is called (auto…) and the default generated directive name

priority = 1

priority if multiple documenters return True from can_document_member

class sage_docbuild.ext.sage_autodoc.**ModuleDocumenter**(**args: Any*)

Bases: *Documenter*

Specialized Documenter subclass for modules.

add_content(*more_content*)

add_directive_header(*sig*)

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

content_indent = ''

indentation by which to indent the directive content

get_module_members()

Get members of target module.

get_object_members(*want_all*)

import_object(*raiseerror=False*)

objtype = 'module'

name by which the directive is called (auto…) and the default generated directive name

option_spec: ClassVar[OptionSpec] = {'deprecated': <function bool_option>, 'exclude-members': <function exclude_members_option>, 'ignore-module-all': <function bool_option>, 'imported-members': <function bool_option>, 'inherited-members': <function inherited_members_option>, 'member-order': <function member_order_option>, 'members': <function members_option>, 'no-index': <function bool_option>, 'no-value': <function bool_option>, 'noindex': <function bool_option>, 'platform': <function identity>, 'private-members': <function members_option>, 'show-inheritance': <function bool_option>, 'special-members': <function members_option>, 'synopsis': <function identity>, 'undoc-members': <function bool_option>}

parse_name()

resolve_name(*modname*, *parents*, *path*, *base*)

sort_members(*documenters*, *order*)

class sage_docbuild.ext.sage_autodoc.**ModuleLevelDocumenter**(*directive: DocumenterBridge*, *name: str*, *indent: str = ''*)

Bases: *Documenter*

Specialized Documenter subclass for objects on module level (functions, classes, data/constants).

resolve_name(*modname*, *parents*, *path*, *base*)

class sage_docbuild.ext.sage_autodoc.**NonDataDescriptorMixin**

Bases: *DataDocumenterMixinBase*

Mixin for AttributeDocumenter to provide the feature for supporting non data-descriptors.

Note

This mix-in must be inherited after other mix-ins. Otherwise, docstring and :value: header will be suppressed unexpectedly.

get_doc()

import_object(*raiseerror=False*)

should_suppress_value_header()

class sage_docbuild.ext.sage_autodoc.**ObjectMember**(*name: str*, *obj: Any*, ***, *docstring: str | None = None*, *class_: Any = None*, *skipped: bool = False*)

Bases: object

A member of object.

This is used for the result of $Documenter.get_module_members()$ to represent each member of the object.

class sage_docbuild.ext.sage_autodoc.**Options**

Bases: dict[str, Any]

A dict/attribute hybrid that returns None on nonexisting keys.

copy()

class sage_docbuild.ext.sage_autodoc.**PropertyDocumenter**(*directive: DocumenterBridge*, *name: str*, *indent: str = ''*)

Bases: *DocstringStripSignatureMixin*, *ClassLevelDocumenter*

Specialized Documenter subclass for properties.

add_directive_header(*sig*)

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

document_members(*all_members=False*)

format_args(***kwargs*)

get_real_modname()

import_object(*raiseerror=False*)

Check the existence of uninitialized instance attribute when failed to import the attribute.

member_order = 60

order if autodoc_member_order is set to 'groupwise'

objtype = 'property'

name by which the directive is called (auto…) and the default generated directive name

priority = 11

priority if multiple documenters return True from can_document_member

class sage_docbuild.ext.sage_autodoc.**RuntimeInstanceAttributeMixin**

Bases: *DataDocumenterMixinBase*

Mixin for AttributeDocumenter to provide the feature for supporting runtime instance attributes (that are defined in __init__() methods with doc-comments).

Example:

class Foo:

def __init__(self):
self.attr = None #: This is a target of this mix-in.

RUNTIME_INSTANCE_ATTRIBUTE

get_doc()

import_object(*raiseerror=False*)

Check the existence of runtime instance attribute after failing to import the attribute.

is_runtime_instance_attribute(*parent*)

Check the subject is an attribute defined in __init__().

is_runtime_instance_attribute_not_commented(*parent*)

Check the subject is an attribute defined in __init__() without comment.

should_suppress_value_header()

class sage_docbuild.ext.sage_autodoc.**SlotsMixin**

Bases: *DataDocumenterMixinBase*

Mixin for AttributeDocumenter to provide the feature for supporting __slots__.

get_doc()

import_object(*raiseerror=False*)

isslotsattribute()

Check the subject is an attribute in __slots__.

should_suppress_value_header()

class sage_docbuild.ext.sage_autodoc.**UninitializedGlobalVariableMixin**

Bases: *DataDocumenterMixinBase*

Mixin for DataDocumenter to provide the feature for supporting uninitialized (type annotation only) global variables.

get_doc()

import_object(*raiseerror=False*)

should_suppress_value_header()

class sage_docbuild.ext.sage_autodoc.**UninitializedInstanceAttributeMixin**

Bases: *DataDocumenterMixinBase*

Mixin for AttributeDocumenter to provide the feature for supporting uninitialized instance attributes (PEP-526 styled, annotation only attributes).

Example:

class Foo:
attr: int #: This is a target of this mix-in.

get_doc()

import_object(*raiseerror=False*)

Check the existence of uninitialized instance attribute when failed to import the attribute.

is_uninitialized_instance_attribute(*parent*)

Check the subject is an annotation only attribute.

should_suppress_value_header()

sage_docbuild.ext.sage_autodoc.**annotation_option**(*arg*)

sage_docbuild.ext.sage_autodoc.**autodoc_attrgetter**(*app*, *obj*, *name*, **defargs*)

Alternative getattr() for types

sage_docbuild.ext.sage_autodoc.**between**(*marker*, *what=None*, *keepempty=False*, *exclude=False*)

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.

sage_docbuild.ext.sage_autodoc.**bool_option**(*arg*)

Used to convert flag options to auto directives. (Instead of directives.flag(), which returns None).

sage_docbuild.ext.sage_autodoc.**class_doc_from_option**(*arg*)

Used to convert the :class-doc-from: option to autoclass directives.

sage_docbuild.ext.sage_autodoc.**cut_lines**(*pre*, *post=0*, *what=None*)

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`.

sage_docbuild.ext.sage_autodoc.**exclude_members_option**(*arg*)

Used to convert the :exclude-members: option.

sage_docbuild.ext.sage_autodoc.**getdoc**(*obj*, **args*, ***kwargs*)

sage_docbuild.ext.sage_autodoc.**identity**(*x*)

sage_docbuild.ext.sage_autodoc.**inherited_members_option**(*arg*)

Used to convert the :inherited-members: option to auto directives.

sage_docbuild.ext.sage_autodoc.**member_order_option**(*arg*)

Used to convert the :member-order: option to auto directives.

sage_docbuild.ext.sage_autodoc.**members_option**(*arg*)

Used to convert the :members: option to auto directives.

sage_docbuild.ext.sage_autodoc.**merge_members_option**(*options*)

Merge :private-members: and :special-members: options to the :members: option.

sage_docbuild.ext.sage_autodoc.**py_ext_sig_re = re.compile('^ ([\\w.]+::)? # explicit module name\n ([\\w.]+\\.)? # module and/or class name(s)\n (\\w+) \\s* # thing name\n (?: \\[\\s*(.*)\\s*])? , re.VERBOSE)**

extended signature RE: with explicit module name separated by "::"

sage_docbuild.ext.sage_autodoc.**setup**(*app*)

2.3 Sage multidocs extension

The goal of this extension is to manage a multi-documentation in Sphinx. To be able to compile Sage's huge documentation in parallel, the documentation is cut into a bunch of independent documentations called "sub-docs", which are compiled separately. There is a master document which points to all the sub-docs. The intersphinx extension ensures that the cross-link between the sub-docs are correctly resolved. However some work is needed to build a global index. This is the goal of the `multidocs` extension.

More precisely this extension ensures the correct merging of

- the todo list if this extension is activated
- the python indexes
- the list of python modules
- the javascript index
- the citations

sage_docbuild.ext.multidocs.**citation_dir**(*app*)

sage_docbuild.ext.multidocs.**fetch_citation**(*app*, *env*)

Fetch the global citation index from the refman to allow for cross references.

sage_docbuild.ext.multidocs.**fix_path_html**(*app*, *pagename*, *templatename*, *ctx*, *event_arg*)

Fix the context so that the files

- search.html
- genindex.html
- py-modindex.html

point to the right place, that is in reference/ instead of reference/subdocument.

sage_docbuild.ext.multidocs.**get_env**(*app*, *curdoc*)

Get the environment of a sub-doc from the pickle

sage_docbuild.ext.multidocs.**get_js_index**(*app*, *curdoc*)

Get the JS index of a sub-doc from the file

sage_docbuild.ext.multidocs.**init_subdoc**(*app*)

Init the merger depending on if we are compiling a sub-doc or the master doc itself.

sage_docbuild.ext.multidocs.**merge_environment**(*app*, *env*)

Merge the following attributes of the sub-docs environment into the main environment:

- titles – Titles
- todo_all_todos – todo's
- indexentries – global python index
- all_docs – needed by the js index
- citations – citations
- domaindata['py']['modules'] – list of python modules

sage_docbuild.ext.multidocs.**merge_js_index**(*app*)

Merge the JS indexes of the sub-docs into the main JS index

sage_docbuild.ext.multidocs.**setup**(*app*)

sage_docbuild.ext.multidocs.**write_citations**(*app*, *citations*)

Pickle the citation in a file.

PYTHON MODULE INDEX

S

INDEX

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

W