sphinxcontrib-wiki Documentation

Release 0.1

Amir Kadivar

Apr 20, 2018

Contents

1 Wiki **3**
- 1.1 Introduction . . . 3
- 1.2 Potentially Asked Questions . . . 4
- 1.3 To Do List . . . 5

2 sphinxcontrib.wiki module **7**

Python Module Index **11**

An extension for sphinx that allows wiki pages to be defined section by section and assembled on demand using two custom directives: `wikisection`, and `wikipage`, respectively. This is particularly useful for code documentation (although arbitrary sphinx documents can use the directives) by allowing external documentation files to be broken down to sections and placed in docstrings for corresponding portions of code.

To get started, install the package via PyPI, follow the docs and sample usage in the tests directory.

CHAPTER 1

Wiki

1.1 Introduction

1.1.1 Basic usage

`sphinxcontrib-wiki` is a Sphinx extension which allows wiki pages to be automatically generated from docstrings. Here is an example:

```
.. wikisection:: example
    :title: Example wiki section title

    Wiki sections must have a body or they are ignored.
```

Wiki pages are generated like this:

```
.. wikipage:: example
    :title: Example wiki section

    Body of a wiki page appears above all its sections.
```

[source: *sphinxcontrib.wiki*]

1.1.2 Section Hierarchy

By default, the structure of the `wikisection` tree within a `wikipage` is determined by module hierarchy, namely:

- Sections within a module are treated as siblings in the order they appear regardless of the depth of the class or function they belong to.
- Sections within sibling modules are treated as siblings in _alphabetic_ order (and not as they are processed as per `autodoc_member_order`).
- Sections within a package are treated as the parent of those in modules immediately within it.

Alternatively, a section can force its parent to a be specific other section, for instance:

```
.. wikisection:: example
    :title: A section with forced parent
    :parent: Example wiki section title

    Mandatory section body.
```

[source: *sphinxcontrib.wiki*]

1.2 Potentially Asked Questions

1.2.1 Section Reuse

It may be desirable to reuse the same section in different wiki pages. This is currently not possible as it requires rethinking the `parent` option. One possibility is to extend the syntax as follows:

```
.. wikisection:: first, second
    :title: Section belonging to many pages
    :parents: first[Parent in first page], second[_default_]
```

Another possibility is to invert the control and let pages decide their hierarchy, requiring some preprocessing on raw sources:

```
.. wikipage:: example
    :tree:
        section1
            section1.1
            section1.2
        section2
```

[source: *sphinxcontrib.wiki*]

1.2.2 Section within Sections

Title elements are not allowed in docstring by docutils or sphinx. Other extensions that require this functionality, e.g. numpy, implement their own preprocessing methods (cf. `source-read` event by sphinx). The possible ways nested sections can be achieved by `sphinxcontrib-wiki` are:

- Separate out the subsection into a `wikisection` of its own and assign its parent to the parent `wikisection`, for instance:

  ```
  .. wikisection:: example
      :title: Parent Section

      Body of parent.

  .. wikisection:: example
      :title: Child Section
      :parent: Parent Section

      Body of child.
  ```

- Use the `rubric` directive which allows adding a title within a directive body but whose information is lost to the table of contents.

[source: *sphinxcontrib.wiki*]

1.2.3 Arbitrary Order of Processing

As long as the order in which sections are encountered is consistent with a DFS of the module hierarchy, i.e up to reorderings of siblings but not violating depth order, there are no problems. The DFS order is what sphinx follows and the only possible chance of variation is in sibling order caused by `autodoc_member_order` (and our output respects this sibling order by default). But if for some reason, this assumption is violated, i.e a section appearing deeper in the module hierarchy is encountered before another section appearing shallower in the module hierarchy, our logic for placing sections in the right place breaks.

[source: *sphinxcontrib.wiki*]

1.2.4 Cycles in parent relationships

If there are any cycles in the parent relationships the entire set of sections within that cycle are swallowed by docutils. Since there is no error raised or warning issued, this extension cannot inform the user either. Checking for cycles requires a quadratic effort to scan the entire parent-child graph which currently seems unnecessary.

[source: *sphinxcontrib.wiki*]

1.2.5 Resolving References

The current implementation stores all sections in the build environment after `doctree-read` and uses them to populate all pages upon `doctree-resolved`. This is needed to allow all sections to be collected before any page is assembled.

[source: *sphinxcontrib.wiki*]

1.3 To Do List

1.3.1 Report

Sphinx API documentation for developing extensions is not friendly to newcomers. Document the available sources (docutils, sphinx, and extensions in sphinxcontrib) and the basics of creating an extension.

[source: *sphinxcontrib.wiki*]

1.3.2 Tree documentation

Document more clearly how the wiki page tree structure is determined by default: it's all in the dots in rst document name, i.e ideally for python packages and potentially confusing otherwise.

[source: *sphinxcontrib.wiki*]

CHAPTER 2

sphinxcontrib.wiki module

Allow wiki pages to be automatically generated from docstrings.

class sphinxcontrib.wiki.**WikiPage**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

Bases: docutils.parsers.rst.Directive

Handler for the `wikipage` directive. Each page has one required argument, its identifier and a required option, its `title`. Optionally, a page can have a body which will be rendered above all its child sections.

has_content = True

option_spec = {'title': <function unchanged>}

optional_arguments = 0

required_arguments = 1

run()

class sphinxcontrib.wiki.**WikiSection**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

Bases: docutils.parsers.rst.Directive

Handler for the `wikisection` directive. Each section has one reqiured argument, the identifier of the wikipage to which it belongs, and one required option `title` which is used by other sections to reference it. Furthermore, each section must have a non-empty body.

By default, the placement of each section, and thus the depth of its title heading, is automatically calculated based on the package hirerarchy. Optionally, the parent of the section in the page tree, which is another section, can be specified as an option, default is `_default_`. Parent resolution is performed as follows:

1. If parent is `_default_`, the parent is the last observed section (as per sphinx-dictated order of `traverse`) whose depth is one less than this section.
2. If parent is the title of another section, that section will be forced to be the parent of this section.
3. If parent is `_none_`, this section is placed in the top level of the corresponding wiki page.

has_content = True

node_class
alias of *wikisection*

option_spec = {'parent': <function unchanged>, 'title': <function unchanged>}

optional_arguments = 0

required_arguments = 1

run()

sphinxcontrib.wiki.**doctree_read**(*app*, *doctree*)
Handler for sphinx's doctree-read event. This is where we remove all wikisection nodes from the doctree and store them in the build environment.

sphinxcontrib.wiki.**doctree_resolved**(*app*, *doctree*, *docname*)
Handler for sphinx's doctree-resolved event. This is where we replace all wikipage nodes based on the stored sections from the build environment and resolve all references.

sphinxcontrib.wiki.**env_merge_info**(*app*, *env*, *docnames*, *other*)
Standard handler for sphinx's env-murge-info event. We need to implement this because we store data in the build environment, cf. sphinx.ext.todo.

sphinxcontrib.wiki.**env_purge_doc**(*app*, *env*, *docname*)
Standard handler for sphinx's env-purge-doc event. We need to implement this because we store data in the build environment, cf. sphinx.ext.todo.

sphinxcontrib.wiki.**setup**(*app*)
Entry point to sphinx. We define:

1. The configuration parameter wiki_enabled, defaulting to False which turns our behavior on and off.
2. Two node types: *wikisection*, and *wikipage*.
3. Two directives: wikisection, and wikipage with handlers *WikiSection* and *WikiPage*.
4. Four hooks, two of which – *doctree_read()* and *doctree_resolved()* – are involved in moving sections from their original place to where the corresponding page is included. The other two – *env_purge_doc()* and *env_merge_info()* – are implemented to make our usage of the build environment parallel-friendly.

class sphinxcontrib.wiki.**wikipage**(*rawsource=''*, **children*, ***attributes*)
Bases: docutils.nodes.General, docutils.nodes.Element

sphinxcontrib.wiki.**wikipage_container**(*env*, *sec_tree*, *page_node*)
Builds a sphinx section corresponding to a wikipage provided the wikisection tree for it is given.

Parameters

- **env** – The build environment (i.e an instance of sphinx.environment.BuildEnvironment).
- **sec_tree** – A list of sphinx sections at the top level of the tree.
- **page_node** – The *wikipage* node as observed in some document.

Returns A sphinx section containing the wiki page.

Return type sphinx.util.compat.nodes.section

sphinxcontrib.wiki.**wikipage_tree**(*app*, *env*, *docname*, *page_node=None*)
Builds a section tree for a given wikipage node by collecting all wikisections from the environment and placing them in the right place.

Parameters

- **app** – The "application", instance of `sphinx.application.Sphinx`.
- **env** – The build environment (i.e an instance of `sphinx.environment.BuildEnvironment`).
- **docname** – The document name where this *wikipage* was found.
- **page_node** – The *wikipage* node as observed in some document.

Returns A list of top level sections.

Return type `list[sphinx.util.compat.nodes.section]`

class sphinxcontrib.wiki.**wikisection**(*rawsource=''*, **children*, ***attributes*)

Bases: `docutils.nodes.section`

sphinxcontrib.wiki.**wikisection_container**(*app*, *env*, *sec_info*)

Builds a sphinx section corresponding to a given `wikisection`.

Parameters

- **app** – The "application", instance of `sphinx.application.Sphinx`.
- **env** – The build environment (i.e an instance of `sphinx.environment.BuildEnvironment`).
- **sec_info** – A dictionary containing stored info about one section, cf. *doctree_read()*.

Returns The sphinx section containing the given wiki section.

Return type `sphinx.util.compat.nodes.section`

Python Module Index

S

sphinxcontrib.wiki, 7

Index

D

doctree_read() (in module sphinxcontrib.wiki), 8
doctree_resolved() (in module sphinxcontrib.wiki), 8

E

env_merge_info() (in module sphinxcontrib.wiki), 8
env_purge_doc() (in module sphinxcontrib.wiki), 8

H

has_content (sphinxcontrib.wiki.WikiPage attribute), 7
has_content (sphinxcontrib.wiki.WikiSection attribute), 7

N

node_class (sphinxcontrib.wiki.WikiSection attribute), 7

O

option_spec (sphinxcontrib.wiki.WikiPage attribute), 7
option_spec (sphinxcontrib.wiki.WikiSection attribute), 8
optional_arguments (sphinxcontrib.wiki.WikiPage attribute), 7
optional_arguments (sphinxcontrib.wiki.WikiSection attribute), 8

R

required_arguments (sphinxcontrib.wiki.WikiPage attribute), 7
required_arguments (sphinxcontrib.wiki.WikiSection attribute), 8
run() (sphinxcontrib.wiki.WikiPage method), 7
run() (sphinxcontrib.wiki.WikiSection method), 8

S

setup() (in module sphinxcontrib.wiki), 8
sphinxcontrib.wiki (module), 7

W

WikiPage (class in sphinxcontrib.wiki), 7
wikipage (class in sphinxcontrib.wiki), 8
wikipage_container() (in module sphinxcontrib.wiki), 8
wikipage_tree() (in module sphinxcontrib.wiki), 8
WikiSection (class in sphinxcontrib.wiki), 7
wikisection (class in sphinxcontrib.wiki), 9
wikisection_container() (in module sphinxcontrib.wiki), 9