

Sphinx and RST syntax guide

(0.9.3)

[PREVIOUS](#) | [NEXT](#) | [MODULES](#) | [INDEX](#)

1. Restructured Text (reST) and Sphinx CheatSheet

Overview

This page describes some of the RST and Sphinx syntax. It is based on resource found at [Sphinx](#) , [Docutils](#) and more generally software documentation written with Sphinx.

This is not an exhaustive description but it should allow you to start and create already nice documentation.

Date: August 14, 2014
Author: Thomas Cokelaer

Contents

- [Restructured Text \(reST\) and Sphinx CheatSheet](#)
 - [Introduction](#)
 - [Text Formatting](#)
 - [Inline markup and special characters \(e.g., bold, italic, verbatim\)](#)
 - [Headings](#)
 - [Internal and External Links](#)
 - [List and bullets](#)
 - [What are directives](#)
 - [Inserting code and Literal blocks](#)
 - [How to include simple code](#)
 - [code-block directive](#)
 - [Include code with the literalinclude directive](#)
 - [Tables](#)

- [Simple tables](#)
- [Multicells tables, first method](#)
- [Multicells table, second method](#)
- [The tabulardown directive](#)
- [The csv-table directive](#)
- [Include other RST files with the toctree directive](#)
- [Python software](#)
 - [Auto-document your python code](#)
 - [python docstrings](#)
- [Images and figures](#)
 - [Include Images](#)
 - [Include a Figure](#)
- [Boxes](#)
 - [Colored boxes: note, seealso, todo and warnings](#)
 - [Topic directive](#)
 - [Sidebar directive](#)
- [Others](#)
 - [Comments](#)
 - [Substitutions](#)
 - [glossary, centered, index, download and field list](#)
 - [Footnote](#)
 - [Citations](#)
 - [More about aliases](#)
 - [Intersphinx](#)
 - [file-wide metadata](#)
 - [metainformation](#)
 - [contents directives](#)
- [Useful extensions](#)
 - [pngmath: Maths and Equations with LaTeX](#)
 - [TODO extension](#)
 - [copybutton](#)

1.1. Introduction

The reStructuredText (RST) syntax provides an easy-to-read, what-you-see-is-what-you-get plaintext markup syntax and parser system. However, you

need to be very precise and stick to some strict rules:

- *like Python, RST syntax is sensitive to indentation !*
- *RST requires blank lines between paragraphs*

This entire document is written with the RST syntax. In the right sidebar, you should find a link **show source**, which shows the RST source code.

1.2. Text Formatting

1.2.1. Inline markup and special characters (e.g., bold, italic, verbatim)

There are a few special characters used to format text. The special character `*` is used to defined bold and italic text as shown in the table below. The backquote character ``` is another special character used to create links to internal or external web pages as you will see in section [Internal and External Links](#).

usage	syntax	HTML rendering
italic	<code>*italic*</code>	<i>italic</i>
bold	<code>**bold**</code>	bold
link	<code>`python` <www.python.org>`_</code>	python
verbatim	<code>``*``</code>	<code>*</code>

The double backquote is used to enter in verbatim mode, which can be used as the escaping character. There are some restrictions about the `*` and ```` syntax. They

- *cannot not be nested,*
- *content may not start or end with whitespace: `* text*` is wrong,*
- *it must be separated from surrounding*

text by non-word characters like a space.

The use of backslash is a work around to second previous restrictions about whitespaces in the following case:

- *this is a *longish* paragraph is correct and gives longish.*
- *this is a long*ish* paragraph is not interpreted as expected. You should use this is a long\ *ish* paragraph to obtain longish paragraph*

In Python docstrings it will be necessary to escape any backslash characters so that they actually reach reStructuredText. The simplest way to do this is to use raw strings by adding the letter `r` in front of the docstring.

Python string	Typical result
<code>r"""*escape* \`with` "\\\""""</code>	<code>*escape* \`with` "\`</code>
<code>"""*escape* \\`with` "\\\\\""""</code>	<code>*escape* \`with` "\`</code>
<code>"""*escape* \`with` "\\\""""</code>	<code>escape with ""</code>

1.2.2. Headings

In order to write a title, you can either underline it or under and overline it. The following examples are correct titles.

```
*****
Title
*****

subtitle
#####

subsubtitle
*****
and so on
```

Two rules:

- *If under and overline are used, their length must be identical*
- *The length of the underline must be at least as long as the title itself*

Normally, there are no heading levels assigned to certain characters as the structure is determined from the succession of headings. However, it is better to stick to the same convention throughout a project. For instance:

- # with overline, for parts
- * with overline, for chapters
- =, for sections
- -, for subsections
- ^, for subsubsections
- ", for paragraphs

1.2.3. Internal and External Links

In Sphinx, you have 3 type of links:

1. External links (http-like)
2. Implicit links to title
3. Explicit links to user-defined label (e.g., to refer to external titles).

1.2.3.1. External links

If you want to create a link to a website, the syntax is

```
`<http://www.python.org/>`_
```

which appear as <http://www.python.org/>. Note the underscore after the final single quote. Since the full name of the link is not always simple or meaningful, you can specify a label (note the space between the label and link name):

```
`Python <http://www.python.org/>`_
```

The rendering is now: [Python.](http://www.python.org/)

Note

If you have an underscore within the label/name, you got to escape it with a ``\`` character.

1.2.3.2. Implicit Links to Titles

All titles are considered as hyperlinks. A link to a title is just its name within quotes and a final underscore:

```
`Internal and External links`_
```

This syntax works only if the title and link are within the same RST file. If this is not the case, then you need to create a label before the title and refer to this new link explicitly, as explained in [Explicit Links](#) section.

1.2.3.3. Explicit Links

You can create explicit links within your RST files. For instance, this document has a label at the top called `rst_tutorial`, which is specified by typing:

```
.. _rst_tutorial:
```

You can refer to this label using two different methods. The first one is:

```
rst_tutorial_
```

The second method use the `ref` role as follows:

```
:ref:`rst_tutorial`
```

With the first method, the link appears as [rst tutorial](#), whereas the second method use the first title's name found after the link. Here, the second method would appear as [Restructured Text \(reST\) and Sphinx CheatSheet](#).

Note that the second method is compulsory if the link is to be found in an external RST file. For instance, the introduction page is an external page with a link called `introduction` at the top of the page. You can jump there by writting `:ref:`introduction``, which appears as: [Why Sphinx and for which users ?](#).

Note

Note that if you use the `ref` role, the final underscore is not required anymore.

1.2.4. List and bullets

The following code:

```
* This is a bulleted list.
* It has two items, the second
  item uses two lines. (note the indentation)

1. This is a numbered list.
2. It has two items too.

#. This is a numbered list.
#. It has two items too.
```

gives:

- This is a bulleted list.
 - It has two items, the second item uses two lines. (note the indentation)
1. This is a numbered list.
 2. It has two items too.
 3. This is a numbered list.
 4. It has two items too.

Note

if two lists are separated by a blank line only, then the two lists are not differentiated as you can see above.

1.3. What are directives

Sphinx and the RST syntax provides directives to include formatted text. As an example, let us consider the **code-block** syntax. It allows to insert code (here HTML) within your document:

```
.. code-block:: html
   :linenos:

   <h1>code block example</h1>
```

Its rendering is:

```
1  <h1>code block example</h1>
```

Here, **code-block** is the name of the directive. **html** is an argument telling that the code is in HTML format, **linenos** is an option telling to insert line number and finally after a blank line is the text to include.

Note that options are tabulated.

1.4. Inserting code and Literal blocks

1.4.1. How to include simple code

This easiest way to insert literal code blocks is to end a paragraph with the special marker made of a double coulumn `::`. Then, the literal block must be indented:

```
This is a simple example::
```

```
import math
print 'import done'
```

or:

```
This is a simple example:
::
```

```
import math
print 'import done'
```

gives:

This is a simple example:

```
import math
print 'import done'
```

1.4.2. code-block directive

By default the syntax of the language is Python, but you can specify the language using the **code-block** directive as follows:

```
.. code-block:: html
   :linenos:

   <h1>code block example</h1>
```

produces

```
1 <h1>code block example</h1>
```

1.4.3. Include code with the literalinclude directive

Then, it is also possible to include the contents of a file as follows:


```
.. literalinclude:: filename
   :linenos:
   :language: python
   :lines: 1, 3-5
   :start-after: 3
   :end-before: 5
```

For instance, the `sample.py` file contents can be printed:

```
1  """ here is a dummy documentation"""
2  import os
3
4  def square(a):
5      """short description of the function square
6
7      Longish explanation: returns the square of a
8
9      :param a: an input argument
10
11      :returns: a*a
12      """
13      return a*a
14
15  assert 4 == square(2)
16
```

1.5. Tables

There are several ways to write tables. Use standard reStructuredText tables as explained here. They work fine in HTML output, however, there are some gotchas when using tables for LaTeX output.

The rendering of the table depends on the CSS/HTML style, not on sphinx itself.

1.5.1. Simple tables

Simple tables can be written as follows:

```
+-----+-----+-----+
| 1      | 2      | 3      |
+-----+-----+-----+
```

which gives:

1	2	3
---	---	---

Size of the cells can be adjusted as follows:

```
+-----+-----+---+
| 1           |      2 | 3 |
+-----+-----+---+
```

renders as follows:

1	2	3
---	---	---

This syntax is quite limited, especially for multi cells/columns.

1.5.2. Multicells tables, first method

A first method is the following syntax:

```
+-----+-----+-----+
| Header 1 | Header 2 | Header 3 |
+=====+=====+=====+
| body row 1 | column 2 | column 3 |
+-----+-----+-----+
| body row 2 | Cells may span columns.|
+-----+-----+-----+
| body row 3 | Cells may | - Cells |
+-----+ span rows. | - contain |
| body row 4 |          | - blocks. |
+-----+-----+-----+
```

gives:

Header 1	Header 2	Header 3
body row 1	column 2	column 3
body row 2	Cells may span columns.	
body row 3	Cells may	▪ Cells
body row 4	span rows.	▪ contain
		▪ blocks.

1.5.3. Multicells table, second method

The previous syntax can be simplified:

```
=====  =====  =====
      Inputs      Output
-----  -----  -----
      A          B      A or B
=====  =====  =====
False    False    False
True     False    True
=====  =====  =====
```

gives:

Inputs	Output

A	B	A or B
False	False	False
True	False	True

Note

table and latex documents are not yet compatible in sphinx, and you should therefore precede them with the a special directive (`.. htmlonly::`)

1.5.4. The `tabularcolumns` directive

The previous examples work fine in HTML output, however there are some gotchas when using tables in LaTeX: the column width is hard to determine correctly automatically. For this reason, the following directive exists:

```
.. tabularcolumns:: column spec
```

This directive gives a “column spec” for the next table occurring in the source file. It can have values like:

```
|1|1|1|
```

which means three left-adjusted (LaTeX syntax). By default, Sphinx uses a table layout with L for every column. This code:

```
.. tabularcolumns:: |1|c|p{5cm}|
```

```
+-----+---+-----+
| simple text | 2 | 3 |
+-----+---+-----+
```

gives

title
simple text 2 3

1.5.5. The `csv-table` directive

Finally, a convenient way to create table is the usage of CSV-like syntax:

```
.. csv-table:: a title
   :header: "name", "firstname", "age"
   :widths: 20, 20, 10
```

```
"Smith", "John", 40
"Smith", "John, Junior", 20
```

that is rendered as follows:

a title

name	firstname	age
Smith	John	40
Smith	John, Junior	20

1.6. Include other RST files with the toctree directive

Sooner or later you will want to structure your project documentation by having several RST files. The **toctree** directive allows you to insert other files within a RST file. The reason to use this directive is that RST does not have facilities to interconnect several documents, or split documents into multiple output files. The **toctree** directive looks like

```
.. toctree::
    :maxdepth: 2
    :numbered:
    :titlesonly:
    :glob:
    :hidden:

    intro.rst
    chapter1.rst
    chapter2.rst
```

It includes 3 RST files and shows a TOC that includes the title found in the RST documents.

Here are a few notes about the different options

- **maxdepth** is used to indicates the depth of the tree.
- **numbered** adds relevant section numbers.
- **titlesonly** adds only the main title of each document
- **glob** can be used to indicate that * and ? characters are used to indicate patterns.
- **hidden** hides the toctree. It can be used to include files that do not need to be shown (e.g. a bibliography).

The glob option works as follows:

```
.. toctree::
   :glob:

   intro*
   recipe/*
   *
```

Note also that the title that appear in the toctree are the file's title. You may want to change this behaviour by changing the toctree as follows:

```
.. toctree::
   :glob:

   Chapter1 description <chapter1>
```

So that the title of this section is more meaningful.

1.7. Python software

Sphinx can be used to create generic documentation, or software documentation dedicated to Python, but not only (can C, C++, ...). Here, we'll focus on Python itself.

1.7.1. Auto-document your python code

Let us suppose you have a python file called *sample.py* with a function called *square*. The function's code is :

```
1 def square(a):
2     """short description of the function square
3
4     longish explanation: returns the square of a
5
6     :param a: an input argument
7
8     :returns: a*a
9     """
10    return a*a
```

Using the **autofunction** :

```
.. currentmodule:: sample
.. autofunction:: square
```

Gives

[source]

square(*a*)

short description of the function square

longish explanation: returns the square of *a*: a^2

Parameters: **a** – an input argument

Returns: $a*a$

Here, we need to specify in which module should be found the function **square**, hence the `.. module::sample` directive. You can use **autoclass** and **automodule** in the same way.

Using the **module** directive also creates an index (see top right of this page) so it is worth specifying more information using platform and synopsis options for example:

```
.. module:: sample
   :platform: Unix, Windows
   :synopsis: sample of documented python code
```

The results will be shown in a module section (link in top right panel).

Note

the directive module should be use only once for a given module.

Warning

the python code must be in the PYTHONPATH.

See also

<http://sphinx.pocoo.org/markup/desc.html>

1.7.2. python docstrings

In a python shell, when you type a statement, it is preceeded by the `>>>` sign. The results are printed without it. For instance:

```
>>> a = 1
1
```

If you want to copy and paste this code, you will get errors since the `>>>` sign is not part of the syntax.

There is a javascript solution to hide it in the [Useful extensions](#) section.

1.8. Images and figures

1.8.1. Include Images

Use:

```
.. image:: stars.jpg
   :width: 200px
   :align: center
   :height: 100px
   :alt: alternate text
```

to put an image



1.8.2. Include a Figure

```
.. figure:: stars.jpg
   :width: 200px
   :align: center
   :height: 100px
   :alt: alternate text
   :figclass: align-center
```

figure are like images but with a caption

and whatever else youwish to add

```
.. code-block:: python
```

```
import image
```

gives

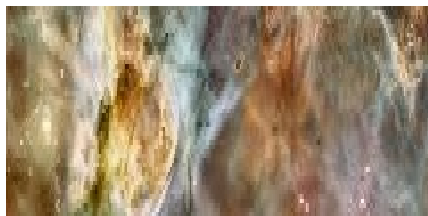


figure are like images but with a caption

and whatever else youwish to add



```
import image
```

The option **figclass** is a CSS class that can be tuned for the final HTML rendering.

1.9. Boxes

1.9.1. Colored boxes: `note`, `seealso`, `todo` and `warnings`


There are simple directives like **seealso** that creates nice colored boxes:



See also

This is a simple **seealso** note.

created using:



```
.. seealso:: This is a simple **seealso** note.
```

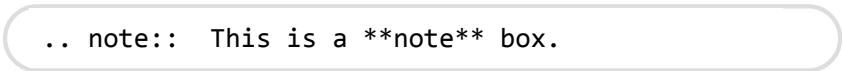
You have also the **note** directive:



Note

This is a **note** box.

with



```
.. note:: This is a **note** box.
```

and the warning directive:



Warning

note the space between the directive and the text

generated with:



```
.. warning:: note the space between the directiv
```

There is another **todo** directive but requires an extension. See [Useful extensions](#)

1.9.2. Topic directive

A **Topic** directive allows to write a title and a text together within a box similarly to the **note** directive.

This code:


```
.. topic:: Your Topic Title
```

Subsequent indented lines comprise the body of the topic, and are interpreted as body elements.

gives

Your Topic Title

Subsequent indented lines comprise the body of the topic, and are interpreted as body elements.

1.9.3. Sidebar directive

It is possible to create sidebar using the following code:

```
.. sidebar:: Sidebar Title
   :subtitle: Optional Sidebar Subtitle
```

Subsequent indented lines comprise the body of the sidebar, and are interpreted as body elements.

1.10. Others

1.10.1. Comments

Comments can be made by adding two dots at the beginning of a line as follows:

```
.. comments
```

1.10.2. Substitutions

Substitutions are defined as follows:

```
.. _Python: http://www.python.org/
```

and to refer to it, use the same syntax as for the internal links: just insert the alias in the text (e.g., Python_, which appears as Python).

A second method is as follows:

```
.. |longtext| replace:: this is a very very long
```

and then insert `|longtext|` wherever required.

1.10.3. glossary, centered, index, download and field list

1.10.3.1. Field list

Whatever: this is handy to create new field
and the following text is indented

```
:Whatever: this is handy to create new field
```

1.10.3.2. glossary

```
.. glossary::
    apical
        at the top of the plant.
```

gives

apical

at the top of the plant.

1.10.3.3. index

```
.. index::
```

1.10.3.4. download

```
:download:`download samplet.py <sample.py>`
```

gives [download sample.py](#)

1.10.3.5. hlist directive

hlist can be use to set a list on several columns.

```
.. hlist::
```

```
.. hlist::
    :columns: 3

    * first item
    * second item
    * 3d item
    * 4th item
    * 5th item
```

- | | | |
|---------------|------------|------------|
| ▪ first item | ▪ 3d item | ▪ 5th item |
| ▪ second item | ▪ 4th item | |

1.10.4. Footnote

For footnotes, use `[#name]_` to mark the footnote location, and add the footnote body at the bottom of the document after a “Footnotes” rubric heading, like so:

```
Some text that requires a footnote [#f1]_ .

.. rubric:: Footnotes

.. [#f1] Text of the first footnote.
```

You can also explicitly number the footnotes (`[1]_`) or use auto-numbered footnotes without names (`[#]_`). Here is an example [\[1\]](#).

1.10.5. Citations

Citation references, like [\[CIT2002\]](#) may be defined at the bottom of the page:

```
.. [CIT2002] A citation
      (as often used in journals).
```


and called as follows:

```
[CIT2002]_
```

1.10.6. More about aliases

Directives can be used within aliases:

```
.. |logo| image:: stars.jpg
   :width: 20pt
   :height: 20pt
```

Using this image alias, you can insert it easily in your text `|logo|`, like this . This is especially useful when dealing with complicated code. For instance, in order to include 2 images within a table, it follows:

```
+-----+-----+-----+
| |logo| | |logo| | |longtext||
+-----+-----+-----+
```



this is a longish
text to include
within a table

g+1 < 4

SEARCH

Go

Enter search
terms or a
module, class or
function name.

This page:

[Show source.](#)

TABLE OF CONTENTS

1. Why Sphinx
and for which
users ?

2. QuickStart

1.
Restructured
Text (reST)
and Sphinx
CheatSheet

2. Example

FAQS

and which is
longer than the
width of the
column.

Changelog

Note

Not easy to get exactly what you want though.

when you create a project, sphinx generates a file containing an index to all the possible links (title, classes, functions, ...).

You can refer to those index only if Sphinx knows where to find this index. This is possible thanks to the **intersphinx** option in your configuration file.

For instance, Python provides such a file, by default Sphinx knows about it. The following code can be found at the end of a typical Sphinx configuration file. Complete it to your needs:

```
# Example configuration for intersphinx: refer to  
intersphinx_mapping = {'http://docs.python.org/'
```

1.10.8. file-wide metadata

when using the following syntax:

```
:fieldname: some contents
```

some special keywords are recognised. For instance, *orphan*, *nocomments*, *tocdepth*.

An example of rendering is the toctree of top of this page.

1.10.8.1. orphan

Sometimes, you have an rst file, that is not included in any rst files (when using include for instance). Yet, there are warnings. If you want to suppress the warnings, include this code in the file:

```
:orphan:
```

There is also `tocdepth` and `nocomments` metadata. See Sphinx homepage.

1.10.9. metainformation

`.. sectionauthor:: name <email>`

Specifies the author of the current section.:

```
.. sectionauthor:: John Smith <js@python.org>
```

By default, this markup isn't reflected in the output in any way, but you can set the configuration value **show_authors** to True to make them produce a paragraph in the output.

1.10.10. contents directives

`.. contents::`

```
.. contents:: a title for the contents
:depth: 2
```

- **depth** indicates the max section depth to be shown in the contents

1.11. Useful extensions

In the special file called **conf.py**, there is a variable called **extensions**. You can add extension in this variable. For instance:

```
extensions = [-
    'easydev.copybutton',
    'sphinx.ext.autodoc',
    'sphinx.ext.autosummary',
    'sphinx.ext.coverage',
    'sphinx.ext.graphviz',
    'sphinx.ext.doctest',
    'sphinx.ext.intersphinx',
    'sphinx.ext.todo',
    'sphinx.ext.coverage',
    'sphinx.ext.pngmath',
    'sphinx.ext.ifconfig',
    'matplotlib.sphinxext.only_directives',
    'matplotlib.sphinxext.plot_directive',
]
```

1.11.1. pngmath: Maths and Equations with LaTeX

The extension to be added is the `pngmath` from sphinx:

```
extensions.append('sphinx.ext.pngmath')
```

In order to include equations or simple Latex code in the text (e.g., $\alpha \leq \beta$) use the following code:

```
:math:`\alpha > \beta`
```

Warning

The *math* markup can be used within RST files (to be parsed by Sphinx) but within your python's docstring. the slashes need to be escaped ! `:math:`\alpha`` should therefore be written `:math:`\\alpha`` or put an "r" before the docstring

Note also, that you can easily include more complex mathematical expressions using the `math` directive:

```
.. math::
```

```
n_{\mathrm{offset}} = \sum_{k=0}^{N-1} s_k r
```

Here is another:

$$n_{\text{offset}} = \sum_{k=0}^{N-1} s_k n_k$$

It seems that there is no limitations to LaTeX usage:

$$s_k^{\text{column}} = \prod_{j=0}^{k-1} d_j, \quad s_k^{\text{row}} = \prod_{j=k+1}^{N-1} d_j.$$

1.11.2. TODO extension

Similarly to the note directive, one can include todo boxes but it requires the *sphinx.ext.todo* extension to be added in the **conf.py** file by adding two lines of code:

```
extensions.append('sphinx.ext.todo')
todo_include_todos=True
```

Todo

a todo box

1.11.3. copybutton

When including Python code with the `>>>` signs, there is a nice extension called `copybutton` that allows to hide the signs hence make a copy/paste possible. I put this extension into the package **easydev**, available on Pypi website. I do not know the origin of this code so sorry if it's yours. If so, let me know so that I can add the author!copyright.

So, if you add the `easydev.extension` into the configuration file

```
extensions.append('easydev.copybutton')
jscopybutton_path = easydev.copybutton.get_copyt

if os.path.isdir('_static')==False:
    os.mkdir('_static')

import shutil
shutil.copy(jscopybutton_path, '_static')

html_static_path = ['_static']
```

Then, you can add a block code (using the `>>>` signs) and you should see a clickable set of characters (`>>>`) in the top right corner to switch on/off the `>>>` signs:

```
>>> a=1
```

>>>

Footnotes

[\[1\]](#) this is a footnote aimed at illustrating the footnote capability.

Bibliography

[\[CIT2002\]](#) A citation (as often used in journals).