
Rest and Sphinx Memo

Release 1.2

Marc Zonzon

Jun 27, 2019

1	Introduction	3
2	ReST – reStructuredText	5
2.1	Structural elements	5
2.1.1	Emacs ReST mode	5
2.1.2	Sectioning	6
2.1.3	Transition	6
2.2	Inline markup	6
2.2.1	ReStructuredText Text Roles.	6
2.3	Lists	7
2.3.1	Bullet list	7
2.3.2	Horizontal lists	7
2.3.3	Enumerated list	8
2.3.4	Definition list	8
2.3.5	Field list	9
2.3.6	Options list	9
2.4	Blocks	9
2.4.1	Literal Block	9
2.4.2	Line blocks	10
2.4.3	Block quote	10
2.4.4	Pull-quote	11
2.4.5	Epigraph and highlights	11
2.4.6	Container	11
2.4.7	Class	11
2.5	Tables	12
2.5.1	Simple tables	12
2.5.2	Grid tables	13
2.5.3	csv table.	14
2.5.4	List Table.	14
2.5.5	LaTeX table rendering.	15
2.6	Cross reference.	16
2.6.1	Hypertext link.	16
2.6.2	Internal document reference.	16
2.6.3	Indirect Hyperlink.	17
2.6.4	Anonymous Hyperlink	17
2.6.5	Implicit Hyperlink Targets	17
2.6.6	Reference in citation style.	18
2.6.7	Embedded URI and Aliases	18
2.6.8	Standalone Hyperlink.	19
2.6.9	Difference between ReST and Sphinx location reference	19
2.7	Explicit Markup	19

2.7.1	Footnote	19
2.7.2	Citation	20
2.8	Rest Directives	21
2.8.1	table of contents	21
2.8.2	image and figure	22
2.8.3	Code blocks	23
2.8.4	Replacement	23
2.8.5	File include	24
2.8.6	Sidebar and Topic	24
2.8.7	Rubric	24
2.8.8	Comment	24
2.8.9	Common options	25
3	Sphinx – Sphinx	27
3.1	Sphinx inline markup	27
3.1.1	Location cross references	28
3.1.2	Automatic labels for sections	28
3.1.3	Cross-referencing documents.	29
3.1.4	Extra cross-reference roles	29
3.1.5	Extensions that define new hyperlinks targets	29
3.1.6	Sphinx Roles.	29
3.1.7	Python Roles.	29
3.2	Sphinx directives	30
3.2.1	Table of contents.	30
3.2.2	Index.	30
3.2.3	Glossary.	31
3.2.4	Note, Warning, Seealso.	31
3.2.5	Selective inclusion.	31
3.3	Defining a css class for some part.	32
3.3.1	Using your new style	33
3.4	Sphinx Source Code.	33
3.4.1	Code highlighting.	33
3.4.2	Source code include.	34
3.4.3	Source code directives.	35
3.4.4	autodoc	35
3.4.5	Using info field lists in Docstrings.	35
3.4.6	Docstring alternate syntax.	36
3.4.7	Google style docstrings.	37
4	Sphinx extensions	41
4.1	todo extension	41
4.2	Sphinx Math	41
4.2.1	Multiline Math	42
4.2.2	Equation Numbers	43
4.3	Graphs with Graphviz	43
4.3.1	Examples	43
5	Sphinx project	45
6	References	47
6.1	Sphinx and Rest References	47
6.2	How to write docstrings	47
6.3	Extending Sphinx	48
7	Indices and tables	49
	Bibliography	51
	Python Module Index	53

Contents:

CHAPTER 1

Introduction

License



Rest and Sphinx Memo by Marc Zonzon
is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](#).
[Source in GitHub](#)

This Memo was written [\[history\]](#) to serve as quick reference for ReST and Sphinx syntax.

Sphinx itself is very well documented, but is very large and cover many aspects, all of them not for daily use. And the Documentation of Rest, while being very rigorous is not of an easy access, at least at my own taste. I use to spend a lot of time to clarify even a tiny aspect, as the documentation is often scattered in many locations.

Moreover Sphinx seems first to be an extension of Rest, but it is not always true and there are some difference of aims, and incompatibilities, between Sphinx and Rest constructs.

This Memo give the constructs that I use the most often, and give cross-references to the manuals for details.

A simple markup language for plain text files.

- *Structural elements*
- *Inline markup*
- *Lists*
- *Blocks*
- *Tables*
- *Cross reference.*
- *Explicit Markup*
- *Rest Directives*

2.1 Structural elements

- *Emacs ReST mode*
- *Sectioning*
- *Transition*

2.1.1 Emacs ReST mode

C-c C-=	Adjust/rotate or promote/demote the decorations
C-- C-c C-=	reverse Adjust
C-c C-a C-d	Display the title decoration hierarchy.
C-- C-c C-r <tab>	shift region left
C-c C-r <tab>	shift region right
C-c C-t C-t	display a table of content to navigate buffer

2.1.2 Sectioning

Titles are under- (and over-)lined (decorated) by `=*-^"~` : . ' #` as below. The exact order of the decoration is not important, the one below is the [Python convention](#).

<pre>#### Part #### ***** Chapter ***** Section =====</pre>	<pre>Subsection ----- Subsubsection ^^^^^^^^^^^^^^ Paragraph """"""""</pre>
---	---

Normal paragraphs are separated by a blank line.

A `=` with overlines is very often preferred to a `*` with overlines for chapters. The previously quoted [Python development guide](#) while advising to use stars uses internally equal character.

[Docutils documentation](#) uses overlined `=` for parts, overlined `-` for chapters, `=` for sections, `-` for subsections, back quotes (```) for subsubsections.

2.1.3 Transition

Any repetition of 4 or more punctuation characters with preceding and trailing blank line is a transition, and looks like this:

2.2 Inline markup

<code>*emphasize*</code>	<i>emphasize</i>
<code>**emphasize strongly**</code>	emphasize strongly
<code>`code`</code>	code
<code>`don't know`</code>	<i>don't know</i>
<code>Asterisk *</code>	Asterisk *
<code>back-quote \`</code>	back-quote ‘
<code>**mark**\ up.</code>	markup.

See [inline markup reference](#).

2.2.1 ReStructuredText Text Roles.

The [ReStructuredText Interpreted Text Roles](#) are valid both for reST and Sphinx processing. They are: `:emphasis:`, `:strong:`, `:literal:`, `:code:`, `:math:`, `:pep-reference:`, `:rfc-reference:`, `:subscript:`, `:superscript:`, `:title-reference:`, `:raw:`. The first three are seldom used because we prefer the shortcuts provided by previous [reST inline markup](#).

The [Custom Interpreted Text Roles](#) which is a reST directive `role`, the tailor the renderer to apply some special formatting. We use it [in Sphinx section](#) to use a special css class for some span of text.

2.3 Lists

- *Bullet list*
- *Horizontal lists*
- *Enumerated list*
- *Definition list*
- *Field list*
- *Options list*

2.3.1 Bullet list

```
- First item with some lengthy
  text wrapping hopefully
  across several lines.

  * We can have subitems
  * separated by a blank line
  * and indented.

- Second item
```

- First item with some lengthy text wrapping hopefully across several lines.
 - We can have subitems
 - separated by a blank line
 - and indented.
- Second item

We can begin each item with `*`, `+`, `-`, `•`, `→`, or `followed by at least one space`, you should keep the indentation of the text of the first line in subsequents lines.

See [bullet list reference](#)

2.3.2 Horizontal lists

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

   * list of
   * short items
   * that should be
   * displayed
   * horizontally
```

- list of
- short items
- that should be
- displayed
- horizontally

hlist is a sphinx extension, not a ReST directive

2.3.3 Enumerated list

```
2. We start with point number 2
#. Automatically numbered item.

a) Point a

    i) first subitem
    ii) second subitem

b) Point b
#) Automatic point c.
```

2. We start with point number 2
3. Automatically numbered item.
 - a) Point a
 - i) first subitem
 - ii) second subitem
 - b) Point b
 - c) Automatic point c.

We can use enumerate with numerals, alphabetic lower case or upper case, roman numerals lower case or upper case.

We can write enumeration followed by a period, a right parenthese, or surrounded by a parentheses; but these punctuation are not kept in the rendering; *rst2html* render `i .`, `i)` or `(i)` as “i.” and Sphinx render them as “a.”.

A list must be separated from previous paragraph by a blank line, in the same way sublists must be separated from items of upper list by a blank line.

Any break of sequence in the source, produces a new list.

See [enumerated list reference](#).

2.3.4 Definition list

```
what
  Definition of "what". We add a few
  words to show the line wrapping.

how
  Definition of "how".

why :
  We define "why" we do it.

  In many paragraphs
```

what Definition of “what”. We add a few words to show the line wrapping.

how Definition of “how”.

why [cause] We define “why” we do it.

In many paragraphs.

With ReST but not Sphinx you can use a classifier after the main term like

```
why : cause
  We define "why" we do it.
```

We may have to escape a markup character to clear ambiguity between a definition list and an other construct like:

```
\(w)
  This is a definition list, not an enumeration.
```

See [definition list reference](#).

2.3.5 Field list

<pre>:Name: Isaac Newton :Long: Here we insert more text in many lines. :Remark: Starts on the next line.</pre>	<p>Name Isaac Newton Long Here we insert more text in many lines. Remark Starts on the next line.</p>
---	--

See [field list reference](#).

2.3.6 Options list

E.g. for listing command line options.

<pre>-v An option -o file Same with value --delta A long option --delta=len Same with value --option-name-is-long description on the next ↪line.</pre>	<pre>-v An option -o file Same with value --delta A long option --delta=len Same with value --option-name-is-long description on the next line.</pre>
---	--

It is nice to align option descriptions, but not mandatory, but at least two spaces must separate an option from the description.

2.4 Blocks

- *Literal Block*
- *Line blocks*
- *Block quote*
- *Pull-quote*
- *Epigraph and highlights*
- *Container*
- *Class*

2.4.1 Literal Block

rest literal blocks

A block which is not interpreted at all is preceded by a paragraph consisting of `::` and a blank line. The block must be indented.

The double `::` is removed from the output.

To use a specific formatting, you can use the *code directive*

<pre>Block one :: **No** interpretation of special characters.</pre>	<p>Block one</p> <p>**No** interpretation of special characters.</p>
--	---

You can also put the `::` at the end of the paragraph preceding the block. When text immediately precedes the `::` the two colons are displayed as “:”, if there is a space before the colons they are removed from the output.

<pre>Block in condensed syntax:: - I'm not a list.</pre>	<p>Block in condensed syntax:</p> <p>- I'm not a list.</p>
<pre>Another block! :: In the text body, indentation is preserved</pre>	<p>Another block!</p> <p>In the text body, indentation is preserved</p>

Warning: *Sphinx* use literal blocks to *highlight source code*, so ****No**** is still written with a bold font by *Sphinx* while being not interpreted by *rst2html*.

To disable *Pygment* decorations in *Sphinx* use a *code block* in text language.

2.4.2 Line blocks

In a line block (*ref*) every line is preceded with `|` and at least one space.

<pre> Line block New line and we are still on the same line Yet a new line</pre>	<p>Line block</p> <p>New line and we are still on the same line</p> <p>Yet a new line</p>
---	---

2.4.3 Block quote

created by ... surrounding paragraph.

```
    Neither from itself nor from another,
    Nor from both,
    Nor without a cause,
```

(continues on next page)

(continued from previous page)

```
Does anything whatever, anywhere arise.
```

```
--Nagarjuna - *Mulamadhyamakakarika*
```

Block quotes (ref) are created by just indenting them more than the surrounding paragraphs.

Neither from itself nor from another, Nor from both, Nor without a cause, Does anything whatever, anywhere arise.

—Nagarjuna - *Mulamadhyamakakarika*

An optional attribution can be set by a line beginning by two or three minus signs flushed left at the level of the quote.

2.4.4 Pull-quote

Pull-quotes (ref) are similar to blockquotes but are *directives*

```
.. pull-quote::
```

```
Just as a solid ...
```

Just as a solid rock is not shaken by the storm, even so the wise are not affected by praise or blame.

2.4.5 Epigraph and highlights

An *epigraph* directive (ref) and an *highlights* directive (ref) are aimed to put a quotation in a distinct font.

don't forget the final s of highlights, or you fall down on the Sphinx code highlighting directive

```
.. highlights::
```

```
With these *highlights* ...
```

With these *highlights* we have completed the ReST blocks.

These three directives are similar in html rendering to *Block quote* but with a *class* of pull-quote, highlights or epigraph that your css may use *but default css does not!*

2.4.6 Container

```
.. container:: myclass
```

```
There is also a general ...
```

There is also a general *container directive* whose unique effect is adding some class name to the block that your css may use. In html this paragraph is enclosed in a

```
<div class="myclass container"> ... </div>
```

2.4.7 Class

```
.. class:: myclass
```

```
The class directive ....
```

```
class reST.myclass
```

The class directive (`ref`) add a class on its content or on the first immediately following non-comment element. The name of the class is normalized by docutil to conform to the regexp: `[a-z] (-? [a-z0-9] +) *`.

Note: While the docutil tool `rst2html` put as expected the previous paragraph in a:

```
<p class="myclass">....</p>
```

Sphinx shadows the class directive, so the previous code will not have the expected result. In Sphinx you have to replace `class` by `rst-class`.

2.5 Tables

- *Simple tables*
- *Grid tables*
- *csv table.*
- *List Table.*
- *LaTeX table rendering.*
 - *Rendering with tabulary.*
 - *Rendering with tabular.*

2.5.1 Simple tables

Code for the examples

```
== ==
aA  bB
cC  dD
== ==

=====
Vokal  Umlaut
=====
aA      äÄ
oO      öÖ
=====

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

=====  =====
1. Hallo  | blah blah blah
          | 1 1 1 1 1 1 1 1
          |
          | blah
12. Here  | blah blah
          | We can wrap the
          | text in source
32. There  **aha**
          =====
```

Chapter 2. ReST – reStructuredText

Simple tables ([ref](#)) are preceded and ended with a sequence of “=” to indicate the columns, e.g:

aA	bB
cC	dD

Headers are indicated by another sequence of “=”, e.g:

Vokal	Umlaut
aA	äÄ
oO	öÖ

Column spans are followed by a sequence of “-” (except for the last header or last row of the table where we must have “=”), e.g:

Inputs		Output
A	B	A or B
False		False
True	False	True
False	True	True
True		True

Simple table cells are treated like a small document on their own up to line breaks, but the first column must contain a single line. e.g:

1. Hallo	blah blah blah blah blah blah blah blah
2. Here	We can wrap the text in source
32. There	aha

2.5.2 Grid tables

Code for example

```
+-----+-----+-----+
| Header | Header with 2 cols |
+=====+=====+=====+
| A      | Lists: | **C** |
+-----+ - aha +-----+
| B::    | - yes | | a block |
|        |      | | of text |
| *hey*  | #. hi | | a break |
+-----+-----+-----+
```

Grid tables ([ref](#)) have a more difficult syntax but can express more complex tables.

Header	Header with 2 cols	
A	Lists: <ul style="list-style-type: none">• aha• yes <ol style="list-style-type: none">1. hi	C
B: <i>*hey*</i>		a block of text a break

You can edit them under emacs with `table.el` (but be careful about conflicts with `rst-mode`) or use *org tables* with `orgtbl-mode` and export to table with `org-table-convert` or `org-table-create-with-table.el` (bound to `C-c ~` in `org-mode`, but not in `orgtbl-mode`)

2.5.3 csv table.

Code

```
.. csv-table:: Balance Sheet
   :header: Description,In,Out,Balance
   :widths: 20, 10, 10, 10
   :stub-columns: 1

   Travel,,230.00,-230.00
   Fees,,400.00,-630.00
   Grant,700.00,,70.00
   Train Fare,,70.00,**0.00**
```

Table 1: Balance Sheet

Description	In	Out	Balance
Travel		230.00	-230.00
Fees		400.00	-630.00
Grant	700.00		70.00
Train Fare		70.00	0.00

The options are explained in the reference: [rst directive: csv-table](#)

You can choose a delimiter with `:delim:` and source an external file with the option `:file:/path/of/the/file`.

2.5.4 List Table.

A list-table ([ref](#)) is a two level list, where the first level is a row and the second one a column list. The number of column must be uniform (*no column span*) but cell may contain structured markup.

```
.. list-table:: Weather forecast
   :header-rows: 1
   :widths: 7 7 7 7 60
   :stub-columns: 1

   * - Day
     - Min Temp
     - Max Temp
     -
     - Summary
```

(continues on next page)



(continued from previous page)

```

* - Monday
  - 11C
  - 22C
  - .. image:: _static/sunny.svg
    :width: 30
  - A clear day with lots of sunshine.
    However, the strong breeze will bring
    down the temperatures.
* - Tuesday
  - 9C
  - 10C
...

```

Table 2: Weather forecast

Day	Min Temp	Max Temp		Summary
Monday	11C	22C		A clear day with lots of sunshine. However, the strong breeze will bring down the temperatures.
Tuesday	9C	10C		Cloudy with rain, across many northern regions. Clear spells across most of Scotland and Northern Ireland, but rain reaching the far northwest.

2.5.5 LaTeX table rendering.

Rendering with *tabulary*.

Sphinx use the latex package *tabulary* to render tables in LaTeX.

Tabulary is an extension of the *tabular* package which calculate the width of columns; it has four new formats specifications: LRCJ for Left (Right, Centered, Justified) column with automatic width.

Sphinx uses by default L, but you can override it with a directive like:

```
.. tabularcolumns:: |L|C|C|R|
```

As examples in this document the *source code directives table* which has a proper Sphinx automatic rendering in tabulary |L|L|, which adapt the column size with a wider left one.

The two first *simple tables* the *csv table* and the *list table* are also rendered in *tabulary* with a proper calculation of table width by latex.

Rendering with *tabular*.

Tables that contain any kind of lists, such as object descriptions, blockquotes, or literal blocks are set by default with the *tabular*. In sphinx prior version 1.6 the *:column:* option for list table is not used for latex, and all columns are of the same size.

You can tailor the rendering by giving *tabularcolumns* directive which uses the *p{width}* column type.

Like this for three uneven columns:

```
.. tabularcolumns::
   |p{0.10\linewidth}|p{0.10\linewidth}|p{0.30\linewidth}|
```

2.6 Cross reference.

- *Hypertext link.*
- *Internal document reference.*
- *Indirect Hyperlink.*
- *Anonymous Hyperlink*
- *Implicit Hyperlink Targets*
- *Reference in citation style.*
- *Embedded URI and Aliases*
- *Standalone Hyperlink.*
- *Difference between ReST and Sphinx location reference*

2.6.1 Hypertext link.

Hypertext links are constituted of a reference and a target.

And there are four types of hyperlink targets:

1. An external hyperlink target is an URI or an email addresses like

```
_Docutils: http://docutils.sourceforge.net/  
_John Lee: john.lee@gmail.com
```

It is possible, but not recommended, to avoid the target reference by using an *anonymous hyperlink*.

2. An *internal document reference* point to some location in the same document.
3. An *indirect hyperlink* has an other hyperlink reference as target.
4. An *implicit hyperlink target* is generated by the *docutil* ReST processor, for all sections of the document.

There exist three ways to write hyperlink references (*ref*)

1. In *Citation Style*
2. Inline with an *Embedded URI* which gives in the same construct both the reference and the target.
3. In a *standalone hyperlink* the text of the target URI is used as reference.

2.6.2 Internal document reference.

To define `label` as label also called *explicit hyperlink target* for any text location internal to a document, precede the text location with:

```
.. _label:  
.. _other label:
```

plus a blank line.

A `:name:` option in any block is also an internal reference target.

You can also use *inline internal targets*

```
which are a _`span of running text` in a paragraph.
```

The ReST way of referencing a label or *hyperlink targets* is:

```
label_ or `other label`_
```

2.6.3 Indirect Hyperlink.

If for the same hyperlink target you want to use a you want to use many references you can use an *indirect hyperlink* or *indirect reference*. With the following indirect references `pocoo`, `Sphinx`, `The manual` and `Documentation` refer to the same place.

Like above don't forget backquotes when there are embedded whitespaces.

```
.. _pocoo: http://sphinx.pocoo.org
.. _Sphinx: pocoo_
.. _The manual: pocoo_
.. _Documentation: `The manual`_
```

Note that if you only want to have multiple link text with the same target you can also use:

```
.. _Sphinx:
.. _The manual:
.. _pocoo: http://sphinx.pocoo.org
```

Multiple adjacent hyperlink references will all point to the same target.

An indirect hyperlink can also be defined inline with an *embedded alias*.

2.6.4 Anonymous Hyperlink

Anonymous hyperlinks are hyperlinks where the target has no label text but begins with double leading underscores, the reference itself ends with two trailing underscores. The target are found by their sequential order in the document. The reference number *n*, reference the target number *n*.

Example:

```
.. __: http://docutils.sourceforge.net/docs/ref/rst/

The `ReST reference manual`__
```

The ReST reference manual

The anonymous hyperlinks make the source text quite obscure, as the association between reference and targets can only be seen by enumerating both. They break easily. Moving a bloc of text with either a target or reference invalidate all anonymous hyperlinks of the document. So it is wise to avoid them.

2.6.5 Implicit Hyperlink Targets

Section titles, footnotes, and citations automatically are *implicit hyperlink targets*. ``Transition`_` produces *Transition*.

In pure ReST syntax you can reference the *Transition* section as *how to draw an horizontal line* with the hyperlink: ``how to draw an horizontal line`_` and the *indirect hyperlink*:

```
.. _how to draw an horizontal line: Transition_
```

You can also use them with an *embedded alias*

<pre> The `Transition` section <Transition> ↪ `__` shows `how to draw an horizontal line`__ in your document.</pre>	The Transition section shows <i>how to draw an horizontal line</i> in your document.
--	---

2.6.6 Reference in citation style.

A link to `Sphinx Home`__ in citation style.

```
.. _Sphinx Home: http://sphinx.pocoo.org
```

A link to [Sphinx Home](#) in citation style (`ref`).

In printed documents the link will be listed similar as a citation, as opposed to HTML documents.

The reference target is composed of words made of alphabetic and numeric characters and characters in the set `[, : _ + -]` without double hyphens, separated by spaces. (`ref`)

The references are equivalents when they differ only by case or number of spaces. The space character class include spaces, horizontal or vertical tabs, newlines, carriage returns, or form feeds.

When the reference has no embedded spaces the backquotes are not necessary:

A link to Sphinx__ in citation style.

If you want to use a *styled reference* you have to use a `ref:replacement`.

2.6.7 Embedded URI and Aliases

Reference: `ref`

You can directly embed an URI, a link target or an internal label in a reference enclosed in `< , >` in the reference.

In the same way than explicit *Indirect Hyperlink*. when we use a target defined elsewhere we have use a trailing underscore.

In the last example we use the label `internal` which is placed before the following section.

In-line versions is `Sphinx Home`__ <code><http://sphinx.pocoo.org>`__</code>	In-line versions is Sphinx Home .
.. _devguide: http://docs.python.org/devguide/ `Python development <devguide_>`__	Python development
`Sphinx main page <Sphinx Home_>`__	Sphinx main page
`Internal target <internal_>`__	<i>Internal target</i>

2.6.8 Standalone Hyperlink.

Reference: `ref`

<pre> We may use URI like http://sphinx.pocoo.org or an email address like project@sphinx.org</pre>	<pre>We may use URI like http://sphinx.pocoo or an email address like project@sphinx.org</pre>
--	--

2.6.9 Difference between ReST and Sphinx location reference

Sphinx has its own *preferred syntax*, it uses:

```
:ref:`displayed text <label>`
```

it is specific to Sphinx and you find it *in the Sphinx section*.

While ReST internal hyperlinks reference a target in the same document, Sphinx allow linking across files of the same project.

These two syntax do not have the same rendering, the text of the target label is used by ReST as default displayed text, while Sphinx syntax either need a reference displayed text or when the target is preceding a section the name of the section is used as default displayed text, the text of the label is never used by Sphinx to display the link.

In this document there is two reference targets before the section *Sidebar and Topic*, the next table show how they are rendered.

<code>`Sidebar`_ versus `Topic`_</code>	<i>Sidebar versus Topic</i>
<code>:ref:`Sidebar` versus :ref:`Topic`</code>	<i>Sidebar and Topic versus Sidebar and Topic</i>

You cannot use the Sphinx `:ref:` syntax to reference *implicit hyperlink targets*. but there is an *autosectionlabel extension* which provides a label for each section across the whole project.

When using the Sphinx syntax it is easier to always define an *explicit target*, which is also is more robust as a rewording of a section title will not invalidate the document.

2.7 Explicit Markup

They all begin with two periods and a white space.

2.7.1 Footnote

ref: `footnotes`

To define a footnote numbered 2 you write it `. . [2]` precedes the definition of the footnote 2. It is referenced by `[2]_`. E.g.

<pre>In the text [2]_. .. [2] In the footnote.</pre>	<p>In the text².</p>
<pre>First automatic [#]_. Another automatic [#]_. .. [#] The first automatic. .. [#] The other automatic.</pre>	<p>First automatic¹. Another automatic³.</p>
<pre>A labeled automatic [#one]_. Another of these [#two]_. .. [#one] footnote. .. [#two] labeled footnotes.</pre>	<p>A labeled automatic⁴. Another of these⁵.</p>
<pre>An autosymbol [*]_. More autosymbol [*]_. .. rubric:: Footnotes .. [*] Footnote in a ↪ *Footnotes* ``rubric`` at end of ↪ document. .. [*] other labeled footnote.</pre>	<p>An autosymbol^{*0}. More autosymbol^{†0}.</p>

Labeled footnotes are always numerics.

2.7.2 Citation

ref: citations and citation references

Citations are identical to footnotes except that they use only non-numeric labels.

<pre>More details are given in [project]_. .. [project] This project began by</pre>	<p>More details are given in [project].</p>
<pre>We can use also links like Project_</pre>	<p>As citation define ordinary target reference We can use also links like <i>Project</i></p>

.. [project] is followed by the definition of the citation It is referenced as [project]__.

Citation labels are **single word** *reference name*.

² In the footnote.

¹ The first automatic.

³ The other automatic.

⁴ footnote.

⁵ labeled footnotes.

⁰ Footnote in a *Footnotes* rubric at end of document.

⁰ other labeled footnote.

In Sphinx, definition and reference can reside in different files.

2.8 Rest Directives

- *table of contents*
- *image and figure*
 - *Images and LaTeX export*
- *Code blocks*
- *Replacement*
- *File include*
- *Sidebar and Topic*
- *Rubric*
- *Comment*
- *Common options*

Directives are a general-purpose extension mechanism. The general syntax is similar to *explicit_markup*:

```
.. <name>:: <argument 1>
           <argument 2>
   :<option 1>: <value>

   <body>
```

The reST directives are detailed in the docutils reference: [reStructuredText Directives](#)

- *table of contents*
- *image and figure*
 - *Images and LaTeX export*
- *Code blocks*
- *Replacement*
- *File include*
- *Sidebar and Topic*
- *Rubric*
- *Comment*
- *Common options*

We have yet see above the directives *Pull-quote* and *Epigraph and highlights*.

2.8.1 table of contents

Create a *table of contents* containing (sub)titles ranging from level 1 to level <number> if you use the `:local:` option the TOC is local to the section where it appears, otherwise it is for the whole file, the title may be empty:

```
.. contents:: `Table of contents`  
:depth: <number>  
:local:
```

2.8.2 image and figure

Images ([ref](#)) are simple pictures, see also [images in the Sphinx documentation](#)

```
.. image:: _static/NeoHittiteSphinx.svg  
:width: 120px  
:alt: Sphinx Neo-Hittite  
:target: https://it.wikipedia.org/wiki/Telepinu_(divinità)
```



You can click on this image to go to the target [Wikipedia \(it\): Telepinu](#).

A **figure** ([ref](#)) add to an image an optional caption and an optional legend.

```
.. figure:: _static/NeoHittiteSphinx.svg  
:width: 120px  
:alt: Sphinx Neo-Hittite  
  
Sphinx Neo-Hittite  
  
Telepinu is an `Hitite <http://en.wikipedia.org/wiki/Hittites>`_  
deity.
```



width 120px

alt Sphinx Hittite

Sphinx Hittite

Telepinu is an [Hitite](#) deity.

Other options are:

- **:scale:** <integer percentage>,
- **:align:** {top|middle|bottom|left|right}

Images and LaTeX export

The reST command `rst2latex` use the width and height of images and figures but the Sphinx LaTeX exporter use also `\includegraphics` to import the figure; but (as far as Sphinx 1.2pre) it does not use the width and height attribute.

To get proper figure size in latex generated by Sphinx you may have either to

- resize the figure before including it,

- use the `:scale:` option that is supported and generates a latex `\scalebox`
- or put a distinct LaTeX code in an raw: `: latex` directive that use something like:

```
\includegraphics[width=60mm, height=40mm]{myfig.png}
```

Latex does not support svg and it has to be converted to eps or pdf, pdf being the only one to support transparency. The conversion can be done with Inkscape, it can be automated as [explained by Johan B. C. Engelen](#). You can also use the [ipe drawing editor](#).

2.8.3 Code blocks

ref: code directive

```
.. code:: <language>
   :linenos:

   <body>
```

is the ReST directive which is called in python *Code highlighting*. or *sourcecode*.

You must use `code-block` or `sourcecode` with Sphinx and the `code` with ReST utilities.

ReST use the same *code highlighting* than Sphinx, look at *Sphinx code highlighting* to learn about the ways to specify it.

2.8.4 Replacement

ReST references: replace directive, unicode directive, date directive, substitution definitions (specification), substitution definition (definition files), Character Entity Sets.

See also: [docutil FAQ: How can I represent esoteric characters?](#).

General replacements:

<pre>This example is stupid .. stupid replace:: not so clever</pre>	<p>This example is not so clever</p>
--	--------------------------------------

One use of replacements is to create styled reference.

If we are not satisfied by a reference like: [more in ReST directives manual](#) that you get with

```
.. _ more in ReST directives manual:
   http://docutils.sourceforge.net/doc...
```

but you want to get: *more in reST directives manual*.

You use the replacement:

```
... want to get: |more-doc|_.

.. |more-doc| replace:: *more in* **reST** *directives manual*
.. _more-doc: http://docutils.sourceforge.net/doc...
```

We also use substitutions to include unicode characters like © with:

```
.. |copy| unicode:: U+000A9 .. COPYRIGHT SIGN
```

If you use Sphinx there are also three predefined substitutions: `|release|`, `|version|`, `|today|`.

2.8.5 File include

To include a ReST file use:

```
.. include:: subdir/incl.rst
```

You can put the file wherever you want the relative paths are interpreted relative to the source directory.

The options: `start-line`, `end-line`, `start-after`, `end-before` as referenced in [reST Directives](#).

If you use `include` with Sphinx, you should exclude the included files from the source file lookup, by setting in `conf.py` the value `exclude_patterns` `<config.html#confval-exclude_patterns>` to a glob pattern in like:

```
exclude_patterns = ["include/**"]
```

For including source code in Sphinx rather use the Sphinx directive [literalinclude](#).

2.8.6 Sidebar and Topic

A [sidebar](#) or a [topic](#) are treated like documents on their own:

```
.. sidebar:: <Title>

    <body>

.. topic:: Topic Title
   :name: mytopic

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

Topic Title

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

2.8.7 Rubric

A [rubric](#) is a title not appearing in the table of contents:

```
.. rubric:: <Title>
```

2.8.8 Comment

Everything starting like a directive with two periods and a space but not a valid construct is a comment. The comment consume all indented text following it. To avoid a confusion with an other constructs, you can let the first line of a block comment empty except the two periods.

When the two dots are not followed by any text, but a blank line, this is an empty comment, that will not consume a following indented block. Empty comments are used to terminate a preceding construct.

```
.. One line comment

..
    A longer comment example
    more comment
```

(continues on next page)

(continued from previous page)

```

    Still in comment

..

    Here is a block-quote,
    not a comment anymore

```

Here is a block-quote, not a comment anymore

2.8.9 Common options

ref: [Directives Common Options](#)

The class options `:class:` and `:name:` are supported by most of the directives.

In the following *topic* and *autre* the `:name:` act as a reference target. Here we can refer to the following block as *say-no-more*.

```

.. topic:: The end
   :class: exceptional
   :name: say-no-more

   A final word.

```

The class render in html as:

```

<div class="exceptional topic" id="say-no-more">
<p class="topic-title first">the end</p>
<p>A final word.</p>
</div>

```

the end

A final word.

- *Sphinx inline markup*
- *Sphinx directives*
- *Defining a css class for some part.*
- *Sphinx Source Code.*

3.1 Sphinx inline markup

- *Location cross references*
- *Automatic labels for sections*
- *Cross-referencing documents.*
- *Extra cross-reference roles*
- *Extensions that define new hyperlinks targets*
- *Sphinx Roles.*
- *Python Roles.*

sphinx reference: [Inline markup](#)

Sphinx inline markup is down through interpreted text roles; they are written `:rolename:`content`..`

There are three types of roles:

- The *ReStructuredText Text Roles*.
- The Sphinx roles that are described in the section *Sphinx Roles*. and the *Sphinx cross references*
- The roles added by [Sphinx domains](#) like the *Python roles* referenced below.

3.1.1 Location cross references

sphinx ref: Cross-referencing arbitrary locations and sphinx role: ref.

We use:

```
:ref:`displayed text <label>`
```

To reference `label` defined in *any* document of the project. It allows linking across files, while the *rest way* is limited to a location in the same file.

If the `label` definition is followed by a section title then `displayed text` can be omitted and will be replaced by the section title.

E.g. the *ReST – reStructuredText Internal document reference* section is preceded with `.. _internal:`, so we have:

<code>:ref:`internal`</code>	<i>Internal document reference.</i>
<code>:ref:`That section <internal>`</code>	<i>That section</i>

We can also use as reference target a *name option* like

<code>:ref:`see this topic <mytopic>`</code>	<i>see this topic</i>
--	-----------------------

See also *Difference between ReST and Sphinx location reference* in the *ReST* chapter.

3.1.2 Automatic labels for sections

Sphinx as an extension `autosectionlabel` that allow to reference each section by its title. Its is similar to *Implicit Hyperlink Targets*, but works across document.

In the doc:*ReST chapter <ReST>* we have used an *Implicit Hyperlink Targets* with:

```
`Transition`_  
`how to draw an horizontal line <Transition>`_
```

we cannot use in the present chapter this ReST way of referencing a target because ReST processor know only one document. but we can use it with the `autosectionlabel` extension with

<pre> The section :ref:`Transition` show :ref:`how to draw an horizontal line ↳<Transition>` in your document.</pre>	<p>The section <i>Transition</i> show <i>how to draw an horizontal line</i> in your document.</p>
---	---

Once you use the `autosectionlabel` extension *new in version 1.4* Sphinx will detect duplicate labels, in contrast with *Implicit Hyperlink Targets* `autolabel` define a new label for each section, so if you have manually put a label before a section title which is identical to the title, it will be detected as *duplicate*.

These duplicate are harmless since they reference the same point. But some title in many document can be identical, you can have many *introduction* or *conclusion* in different parts. The duplicate may be problematic as any one can be matched by a reference. To disambiguate the labels there is an configuration option *beginning version 1.6* `autosectionlabel_prefix_document` which prefix the automatic labels with with the name of the document it is in, followed by a colon.

With this setting instead of `:ref:`Transition`` you have to use `:ref:`ReST:Transition`` it avoid a potential ambiguity with a *Transition* paragraph in an other document, and has the additional benefit to avoid also all ambiguities with explicit labels in your documents.

3.1.3 Cross-referencing documents.

sphinx ref: Cross-referencing documents

In Sphinx it is possible to reference a document as follows

<code>:doc:`ReST`</code>	<i>ReST – reStructuredText</i>
--------------------------	--------------------------------

3.1.4 Extra cross-reference roles

Many are described in the [sphinx ref:Cross-referencing other items of interest](#).

To reference a Python Enhancement Proposal use `:pep:`, for a Request for Comments `:rfc:`

3.1.5 Extensions that define new hyperlinks targets

- The [intersphinx extension](#) generates automatic links to the documentation in other projects for objects not in your own project. It interprets the references to [roles](#)

To configure it, give in `conf.py` a dictionary like:

```
intersphinx_mapping = {
    'python': ('http://docs.python.org/3', None)}
```

- The extension [extlinks](#) help when you have many links pointing to the same site. I use it for the previous reference with the code `:sphinx:`extlinks <ext/extlinks.html>`` and the configuration:

```
extlinks = {'sphinx': ('http://sphinx.pocoo.org/en/latest/%s', 'Sphinx: ')}
```

3.1.6 Sphinx Roles.

The Sphinx roles are described in [Sphinx: Inline markup](#) and in the specific domains, e.g. [python roles](#).

Some common markup are:

<code>:abbr:`RFC (request for comments)`</code>	<i>RFC (request for comments)</i>
<code>:file:`/etc/profile`</code>	<i>/etc/profile</i>
<code>:manpage:`ls(1)`</code>	<i>ls(1)</i>
<code>:regexp:`^[a-z]*.[0-9]`</code>	<i>^[a-z]*.[0-9]</i>
<code>:samp:`cp {file} {target}`</code>	<i>cp file target</i>

3.1.7 Python Roles.

role	reference
<code>:py:mod:</code>	module
<code>:py:func:</code>	function
<code>:py:data:</code>	module-level variable.
<code>:py:const:</code>	constant
<code>:py:class:</code>	class ¹
<code>:py:meth:</code>	method ¹²
<code>:py:attr:</code>	data attribute of an object
<code>:py:exc:</code>	exception ¹

You may supply an explicit title and reference target: `:role:`title` <target>``.

3.2 Sphinx directives

Sphinx includes its own [directives](#), which are not available in the docutils builders.

- *Table of contents.*
- *Index.*
- *Glossary.*
- *Note, Warning, Seealso.*
- *Selective inclusion.*

3.2.1 Table of contents.

```
.. toctree::  
    :maxdepth: <depth>  
    :glob:  
  
    <file 1 without file name extension>  
    <file 2 without file name extension>
```

The `toctree` directive create a table of contents across files.

A `glob` option enables to use wildcards in the filenames, e.g. `/comp/*` means all files under the directory `comp`.

Relative names are relative to the document the directive occurs in, absolute names are relative to the source directory.

The depth can be further restricted per file by inserting the following *Field list* type element in the very first line of the file:

```
:tocdepth: <depth>
```

See [Sphinx: Toc tree](#) for other options.

To get a table of content *inside* a file, use the *reST table of contents*

3.2.2 Index.

Entries in the [index](#) are created automatically from all information units like functions, classes or attributes but those with a `:noindex:` option. Explicit manual entries are made as:

```
.. index:: <entry 1>, <entry 2>, !<entry 3> ...  
    single: <entry>; <sub-entry>  
    pair: <1st part>; <2nd part>  
    triple: <1st part>; <2nd part>; <3rd part>
```

The first two versions create single (sub-)entries, while *pair* creates two entries “<1st part>; <2nd part>” and “<2nd part>; <1st part>”; and *triple* makes three entries.

With the exclamation mark, the `<entry 3>` is the main entry for this term and is put in bold.

You can also use the keywords *see* and *seealso* with `see: foo bar` or `seealso: bar foo`.

¹ Class, methods, exceptions may be dotted names.

² The role text should include the type name and the method name

3.2.3 Glossary.

A `glossary` is a *Definition list*:

```
.. glossary::
   :sorted:

   name1
   name2
      definition of both name1 and name2
```

3.2.4 Note, Warning, Seealso.

They are paragraph-level markups

Code

```
.. note:: <text>

.. warning:: <text>

.. seealso::

   <reST definition list>
```

Note: This is a note.

Warning: This is a warning.

See also:

Apples A kind of fruit.

3.2.5 Selective inclusion.

A block may be included depending of the presence of some tag ([Sphinx ref](#)):

```
..only:: <expression>
```

The expression is made of *tags* combined in boolean expressions like `html` and `draft`.

The format and the name of the current builder is set as predefined tag, if needed it can be prefixed to differentiate format and builder, like `format_html` or `builder_html`

You can define tags via the `-t` command-line option of `sphinx-build`.

In the configuration file you can use `tags.has('tag')` to query, `tags.add('tag')` and `tags.remove('tag')` to change.

An alternative is the `ifconfig` directive ([Sphinx ref](#)) from the `sphinx.ext.ifconfig` extension:

```
.. ifconfig:: <Python expression>
```

To evaluate the expression all variables registered from `conf.py` are availables, to add a config value use the `setup` function in `conf.py`:

```
def setup(app):
    app.add_config_value('newconf', 'default', True)
```

the third parameter should always be `True`.

3.3 Defining a css class for some part.

There is at least three ways of doing it:

Rendered result

An example of red text.

Here the full block is red.

An undecorated paragraph.

This paragraph too is is red.

Big warning

Big warning text is red.

```
.. role:: red

An example of :red:`red text` .

.. container:: red

    Here the full block is red.

An undecorated paragraph.

.. class:: red

This paragraph too is is red.

.. admonition:: Big warning
   :class: red

    Big warning text is red.
```

After applying *rst2html* you get:

```
<p>An example of <span class="red">red text</span>.</p>
<div class="red container">
Here the full block of test is red.</div>
<p>An undecorated paragraph.</p>
<p class="red">This paragraph too is is red.</p>
<div class="red admonition">
<p class="first admonition-title">Big warning</p>
<p class="last">Big warning text is red.</p>
```

Here I have taken the *admonition* directive as example but any directive that allow the *:class:* option would do.

As it generates a *span* the *role* directive is the only one that allow to apply your style to a part of a paragraph.

The *class* works as expected with *rest2html*, but directive fail with **Sphinx**. You have to replace it with

```
.. rst-class:: red
```

This paragraph too is is red.

Sphinx use `rst-class` to replace the ReSt *class* directive that is shadowed by Sphinx. This is *only* stated in a small footnote of Sphinx reSt Primer.

3.3.1 Using your new style

To use your new class you need a css style like:

```
.red {
  color: red;
}
```

You put it in a stylesheet, to give it's location:

- With `rst2html` you must specify the stylesheet's location with a `--stylesheet` (for a URL) or `--stylesheet-path` for a local file.
- With Sphinx a flexible solution is to add your own css file in the `_static` directory and give its location with a template that you put in the `_template` directory. You can use a file `layout.html` wich extend the original template of the same name:

```
{% extends "!layout.html" %}
{% set css_files = css_files + ["_static/style.css"] %}
```

For more details refer to [Sphinx: Templating](#).

3.4 Sphinx Source Code.

- *Code highlighting.*
- *Source code include.*
- *Source code directives.*
- *autodoc*
- *Using info field lists in Docstrings.*
- *Docstring alternate syntax.*
- *Google style docstrings.*
 - *How napoleon transform my docstrings.*

3.4.1 Code highlighting.

Sphinx ref: showing code

The code blocks are highlighted by sphinx, there is a default language of `Python` that can be changed in the configuration, by setting the configuration option `highlight_language`.

The default **Highlighting language** used by `Pygment` in *Literal Blocks* is set for following snippets of code examples by:

```
.. highlight:: <language>
   :linenothreshold: <number>
```

The option language may be any language supported by a [Pygment lexer](#).

The additional linenothreshold option switches on line numbering for blocks of size beyond <number> lines.

When using Sphinx you can specify the highlighting in a single literal block:

```
.. code-block:: <language>
   :linenos:

   <body>
```

The linenos option switches on line numbering. You can also use some options that are described for the [Source code include](#)., namely linenos, lineno-start, caption, name, dedent.

When using base ReST parser use instead [code keyword](#).

3.4.2 Source code include.

Source code is included in Sphinx with the directive [literalinclude](#).

To include the source file `example.py` as a literal block use:

```
.. literalinclude:: code/example.py
   :linenos:
```

```
print("Hello {0}!".format("World"))
```

There are more options:

```
.. literalinclude:: code/example.py
   :caption: Example of code
   :name: example.py
   :language: python
   :dedent: 4
   :linenos:
   :lines: 1,3,5-10,20-
   :emphasize-line: 4,5
```

caption is the displayed title before the block, name is the [reST name option](#) used as an [internal reference](#) target.

dedent strip left whitespaces, linenos add a line numbering alternatively lineno-start begin the numbering at the given number.

The options language and linenos set the highlighting to <language> and enables line numbers respectively.

You can select lines by the lines option or by start-after: <string> and/or end-before: <string> (<string>s are not quoted).

When emphasize the fourth and fifth *selected* line, which in the above example are the lines 5 and 10 of the source.

If it is a Python module, instead of selecting by lines you can select a class function or method to include by using the option pyobject:

```
.. literalinclude:: code/example.py
   :pyobject: MyClass.some_method
```

For including a ReST source file use the [rest directive include](#).

3.4.3 Source code directives.

There are very powerful directives in Sphinx for documenting source code, each programming language has a [specific domains](#).

The following markups are related to [documenting python source code](#).

<code>.. module:: name</code>	Marks the beginning of the description of a module
<code>.. currentmodule:: name</code>	The classes, functions etc. documented from here are in the given module
<code>.. exception:: name[(signature)]</code>	Describes an exception class.
<code>.. class:: name[(signature)]</code>	Describes a class. ³
<code>.. attribute:: name</code>	Describes an object data attribute.
<code>.. method:: name(signature)</code>	Describes an object method.
<code>.. staticmethod:: name(signature)</code>	Describes a static method.
<code>.. classmethod:: name(signature)</code>	Describes a class method.
<code>.. decorator:: name(signature)</code>	Describes a class method.
<code>.. classmethod:: name(signature)</code>	Describes a class method.

3.4.4 autodoc

There is an [autodoc](#) version of the [source code directives](#) which include documentation from docstrings:

- `automodule`, `autoclass`, `autoexception`. Document a module, class or exception. They insert the docstring of the object itself; if you add a `:members:` option followed by a specific list of members, they will be included in the documentation. An empty list of members includes all members.

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

- `autofunction`, `autodata`, `automethod`, `autoattribute` are used to document the respective type of object. `autodata` and `autoattribute` support an annotation option that will show not only the name but the representation of the object.

3.4.5 Using info field lists in Docstrings.

Code for example

```
.. function:: divide( i, j)

    divide two numbers

    :param i: numerator
    :type i: int
    :param j: denominator
    :type j: int
    :return: quotient
    :rtype: integer
    :raises: :exc:`ZeroDivisionError`
```

³ Methods and attributes belonging to the class should be placed in this directive's body.

Inside Python object description directives the following fields are recognized: `param`, `arg`, `key`, `type`, `raises`, `raise`, `except`, `exception`, `var`, `ivar`, `cvar`, `returns`, `return`, `rtype`.

Sphinx.**divide** (*i*, *j*)
divide two numbers

Parameters

- **i** (*int*) – numerator
- **j** – denominator

Returns quotient

Return type integer

Raises `ZeroDivisionError`

```
def divide(self, i, j):
    """Divide two numbers.

    This function is to show how the docstring look, using pure Sphinx syntax,
    with explicit fields.

    :param i: numerator
    :type i: int
    :param j: denominator
    :type j: int
    :return: quotient
    :rtype: integer
    :raises: :exc:`ZeroDivisionError`
    """
```

code.divide.**divide** (*self*, *i*, *j*)
Divide two numbers.

This function is to show how the docstring look, using pure Sphinx syntax, with explicit fields.

Parameters

- **i** (*int*) – numerator
- **j** (*int*) – denominator

Returns quotient

Return type integer

Raises `ZeroDivisionError`

3.4.6 Docstring alternate syntax.

For docstrings you can use the *previous fields* or the alternate syntax that is recommended by Openalea Project. :

```
def example(arg1, arg2):
    """Docstring example.

    :Parameters:
    - `arg1` (float) - the first value
    - `arg2` (float) - the second value

    :Returns:
    arg1/arg2

    :Returns Type:
    float
```

(continues on next page)

(continued from previous page)

```

:Examples:

>>> import template
>>> a = MainClass()
>>> a.example(6,2)
1.5

.. note:: can be useful to emphasize
         important features.
.. seealso:: :py:exc:`ZeroDivisionError`
.. warning:: arg2 must be non-zero.
.. todo:: check that arg2 is non zero.
"""

```

code.docstring.**example**(arg1, arg2)

Docstring example.

Parameters

- *arg1* (float) - the first value
- *arg2* (float) - the second value

Returns arg1/arg2

Returns Type float

Examples

```

>>> import template
>>> a = MainClass()
>>> a.example(6,2)
1.5

```

Note: can be useful to emphasize important features.

See also:

`ZeroDivisionError`

Warning: arg2 must be non-zero.

Todo: check that arg2 is non zero.

3.4.7 Google style docstrings.

Using Restructured texts in your python code, is quite cumbersome, it produces an easily readable documentation, but cryptic source code.

If we want to follow the style of [PEP257](#) we can use the [Google Python style guide](#) also recommended by the [Khan Academy Style Guide](#).

[NumPy style guide](#) Alternatively there is a [NumPy/SciPy style of documentation](#).

But they produces docstrings that are not recognized by Sphinx, and not decorated in html output.

The extension `napoleon` preprocess NumPy and Google style docstrings and converts them to reStructuredText before Sphinx parse the source code. You need to [configure it in your conf.py file](#).

```
def example(arg1, arg2):
    """Division of two reals.

    This function is only to show how we can format docstrings using
    Google Style Guide.

    Args:
        - arg1 (float): The first value.
        - arg2 (float): The second value, cannot be Nul.

    Returns:
        float: Division arg1/arg2.

    Raises:
        ZeroDivisionError: if arg2 is zero.

    Example:
        >>> import template
        >>> a = MainClass()
        >>> a.example(6,2)
        1.5

    Todo:
        Check that arg2 is non zero, and throw an exception.
    """
```

`code.gstyle_docstring.example(arg1, arg2)`

Division of two reals.

This function is only to show how we can format docstrings using Google Style Guide.

Args:

- arg1 (float): The first value.
- arg2 (float): The second value, cannot be Nul.

Returns: float: Division arg1/arg2.

Raises: ZeroDivisionError: if arg2 is zero.

Example:

```
>>> import template
>>> a = MainClass()
>>> a.example(6,2)
1.5
```

Todo: Check that arg2 is non zero, and throw an exception.

Look [here](#) for a larger example.

How *napoleon* transform my docstrings.

We may be curious to know what exactly `napoleon` generates. It can allow us to fix error in the output by changing our docstring or adding Sphinx fields.

We can call `napoleon` from python to learn what it generates:

```
>>> docstring="""
... Division of two reals.
```

(continues on next page)

(continued from previous page)

```

...
... """
>>> from sphinx.ext.napoleon import Config
>>> from sphinx.ext.napoleon.docstring import GoogleDocstring
>>> config = Config(napoleon_use_param=True, napoleon_use_rtype=True)
>>> print(GoogleDocstring(docstring), config)

```

And we have the decorated result:

```

Division of two reals.

This function is only to show how we can format docstrings using
Google Style Guide.

:param - arg1: The first value.
:type - arg1: float
:param - arg2: The second value, cannot be Nul.
:type - arg2: float

:returns: Division arg1/arg2.
:rtype: float

:raises: :exc:`ZeroDivisionError` -- if arg2 is zero.

.. rubric:: Example

>>> import template
>>> a = MainClass()
>>> a.example(6,2)
1.5

.. todo:: Check that arg2 is non zero, and throw an exception.

```

If you want to experiment with output of *napoleon* you can look at the parameters of `GoogleDocstring` in the [source code](#)

We have described in the previous section the extensions: *intersphinx*, *extlinks*, *ifconfig*, *napoleon*.

4.1 todo extension

The extension `sphinx.ext.todo` allow to include todo blocks like

```
.. todo::  
  
    We need to achieve:  
  
    .. include:: include/feature.rst
```

An other directive `todo` is replaced by a list of all todo directives in the whole documentation.

These blocks are by default excluded but can be included by setting to `True` the configuration variable `todo_include_todos`.

You can either set it in the `conf.py` file or trigger it by adding the option to `sphinx-build`. An easy way is through the Make process by doing:

```
$ make -k html SPHINXOPTS="-D todo_include_todos=1"
```

4.2 Sphinx Math

There are three mathematical typesetting Sphinx extensions *imgmath*, *mathjax*, and *jsmath*.

The extension *imgmath* use LaTeX and `dvipng` or `dvisvgm` to render math into PNG or SVG images. You need to install one of these utilities on the machine where the doc is built.

To enable the extension, the following line has to appear in `conf.py`:

```
extensions = ['sphinx.ext.imgmath']
```

You then can type standard LaTeX math expressions, either inline:

```
:math:<LaTeX math expression>
```

or in display mode:

```
.. math::  
    <LaTeX math expressions>
```

The second version is also available for a one line expression:

```
.. math:: <1 Line LaTeX math expression>
```

Code for example

```
Pythagoras :math:`a^2+b^2=c^2`  
  
.. math:: \sum_{n=0}^N x_n = y
```

E.g:

Pythagoras $a^2 + b^2 = c^2$

$$\sum_{n=0}^N x_n = y$$

[Mathjax](#) and its predecessor [jsmath](#) render math through javascript.

4.2.1 Multiline Math

Code for example

```
.. math::  
  
    a+b = c  
  
    b = x_n  
  
    a &= y_n\\  
    &= c-b
```

Sphinx Built-in Mechanism

Several lines of math expressions can be entered by leaving a blank line between them. In addition there is something like an `align` environment syntax if lines are not separated by a blank line.

$$\begin{aligned} a + b &= c \\ b &= x_n \\ a &= y_n \\ &= c - b \end{aligned}$$

Explicit LaTeX with amsmath mechanism

If the option `nowrap` is specified then the full LaTeX code (including the `math`-environment) has to be given. We can assume that the `amsmath` package is loaded. This is not limited to math typesetting, any LaTeX construct can be rendered in this way.


```
.. math:: \[a = b\]
:nowrap:
```

or equivalently

```
.. math::
:nowrap:

\[a = b\]
```

$a = b$

or equivalently

$a = b$

4.2.2 Equation Numbers

Equations are labeled with the `label` option and referred to using:

```
:eq:<label>
```

Code for example

```
.. math:: a^2 + b^2 = c^2
:label: pythag

See equation :eq:`pythag`.
```

E.g:

$$a^2 + b^2 = c^2 \tag{4.1}$$

See equation (4.1).

4.3 Graphs with Graphviz

The [Graphviz graph drawing Sphinx extension](#) is provided in Sphinx distribution.

To enable the extension we have to add it to the `extensions` list in `conf.py`:

```
extensions = ['sphinx.ext.graphviz']
```

It uses directly the `dot` command to process [DOT language](#).

4.3.1 Examples

graph

Undirected:

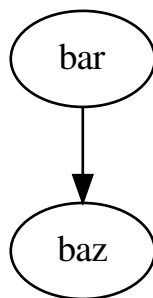
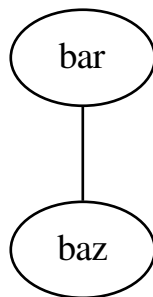
```
.. graph:: foo

    "bar" -- "baz";
```

Directed:

```
.. digraph:: foo

    "bar" -> "baz";
```



CHAPTER 5

Sphinx project

To start a Sphinx project use the interactive `sphinx-quickstart` command ([Sphinx ref](#)). This will ask you all the necessary questions. You can use a `Makefile` to build the documentation.

Customization is done in the file `conf.py` ([Sphinx ref](#)) and the `Makefile` ([Sphinx ref](#)).

The configuration of the extensions go also in `conf.py`.

See [Sphinx description of configuration options](#).

6.1 Sphinx and Rest References

- This doc began as a fork of a 2011 version of [Cristoph Reller reST Memo](#) which is now part of his [Programming Notes](#). I adapted it according to my needs, and they have largely diverged now, but it still inherits from some content and layout.
- [Sphinx documentation](#)
- [Sphinx reStructuredText Primer](#)
- [Documenting Your Project Using Sphinx](#) from an example pypi project's
- [Thomas Cokelaer Openalea project: How to use sphinx ?](#) and [Sphinx and RST syntax guide](#).
- The [ReStructuredText Documentation](#)
 - [Docutil reStructuredText Primer](#) you may prefer the python the *Sphinx* nicely formatted documentation cited above, also available *with a distinct layout* as [docs.python: reStructuredText Primer](#)
 - [Quick reStructuredText](#)
 - [reStructuredText Markup Specification](#)
 - [reST Directives](#)
 - [Interpreted Text Roles](#)
 - [ReStructuredText Demonstration](#)
- [Emacs Support for reStructuredText](#)
- [Documenting Python in the Python Developer's Guide](#)
- [sampledoc tutorial from matplotlib a python 2D plotting library](#).
- [rst2pdf](#) is a tool for transforming reStructuredText to PDF using ReportLab. It supports Sphinx formatting.
- [Epydoc reST markup](#)

6.2 How to write docstrings

- Look at examples in [Official list of projects using Sphinx](#)

- [Documenting Python](#) use Sphinx in function definitions it prefers to the pure Sphinx Syntax the [Google style guide](#) that is used in [Full Code Example](#)
- [OpenAlea](#) has a nice [comparaison of three ways of filling the docstring](#). which compare *Pure sphinx code*, *restructuredText* and *Sphinx, Numpy style*. But it predates the *Napoleon* extension, so the cause of rejecting Numpy may be no longer valid. The source is [template.py](#)
- [Google style guide](#).
- [NumPy style guide](#).
- Sources of [mongo python driver](#) are also a good example

6.3 Extending Sphinx

- [Sphinx Tutorial: Writing a simple extension](#).
- [Creating Custom Link Roles](#).
- [Defining Custom Roles in Sphinx](#) a Sphinx blog post by Doug Hellmann
- [Creating Interpreted Text Roles](#) from docutils project.
- [Creating reStructuredText Directives](#) from docutils project.

CHAPTER 7

Indices and tables

- `genindex`
- `modindex`
- `search`

Bibliography

[history] This Memo is a fork of a previous version of [Cristoph Reller reST Memo](#) it has been deeply modified and largely extended, as Cristoph Reller also modified his memo, they have few in common.

[project] This project began by . . .

r

reST, [??](#)

s

Sphinx, [27](#)

A

- alias
 - embedded, 18
 - hyperlink, 17
- amsmath, 42
- anonymous
 - hyperlink, 17
- attribute
 - python directive, 34
- autodoc
 - directive, 35
- autosectionlabel
 - sphinx extension, 28

B

- block, 9
 - container, 11
 - line, 10
 - literal, 9, 33
 - quotes, 10
- blockquote, 10
 - epigraph, 11
 - highlights, 11
- bullet list, 7

C

- citation, 20
- class
 - directive, 11, 32
 - python directive, 34
- classmethod
 - python directive, 34
- code, 6
 - directive, 23
- code-block
 - directive, 23, 34
- coding style
 - google, 37
 - numpy, 37
- comment, 24
- conf.py, 45
- config value, 31
- configuration

- sphinx, 45
- container
 - directive, 11, 32
- contents
 - directive, 21
- cross
 - reference, 16
- cross-reference
 - role, 29
 - sphinx, 27
- cross-reference document
 - sphinx, 28
- csv
 - directive table, 14
- currentmodule
 - python directive, 34

D

- date
 - directive, 23
- decorator
 - python directive, 34
- definition list, 8
- directive, 21
 - autodoc, 35
 - class, 11, 32
 - code, 23
 - code-block, 23, 34
 - container, 11, 32
 - contents, 21
 - date, 23
 - epigraph, 11
 - figure, 22
 - glossary, 30
 - highlight, 33
 - highlights, 11
 - ifconfig, 31
 - image, 22, 23
 - include, 23
 - index, 30
 - literalinclude, 34
 - module, 34
 - note, 31
 - only, 31

- pull-quote, 11
- replace, 23
- role, 32
- rst-class, 32
- seealso, 31
- source code, 34
- sourcecode, 23
- sphinx, 30
- table csv, 14
- table list, 14
- toctree, 30
- unicode, 23
- warning, 31

directives

- rest, 21

docstring, 35

document cross-reference

- sphinx, 28

E

emacs

- mode, 5

embedded

- alias, 18
- hyperlink, 18
- reference, 18

emphasize, 6

enumerated list, 7

exception

- python directive, 34

extension

- autosectionlabel, 28
- extlink, 29
- graphviz, 43
- ifconfig, 31
- intersphinx, 29
- napoleon, 37
- pngmath, 41

extlink

- extension, 29

F

field

- list, 9

figure

- directive, 22
- latex, 22

file include, 23

footnote, 19

G

glossary

- directive, 30

Graphviz, 43

graphviz

- extension, 43

grid

- table, 13

H

highlight

- directive, 33

horizontal list, 7

hyperlink, 16

- alias, 17
- anonymous, 17
- embedded, 18
- implicit target, 17
- indirect, 17
- standalone, 18
- target, 29

hypertext

- link, 16
- target, 16

I

ifconfig

- directive, 31
- extension, 31

image

- directive, 22, 23
- latex, 22
- replacement, 23

implicit

- hyperlink, 17

in-line

- reference, 18

in-line markup

- sphinx, 27

include

- directive, 23
- selective, 31
- source code, 34

index

- directive, 30
- pair, 30
- see, 30
- seealso, 30
- single, 30
- triple, 30

indirect

- hyperlink, 17
- reference, 16

info fields, 35

inline markup, 6

- code, 6
- emphasize, 6
- strong, 6

internal

- reference, 16

intersphinx

- extension, 29

L

label

- reference, 16

latex

- table, 15
 - tabular, 15
 - tabulary, 15
- link
 - hypertext, 16
- list, 6
 - bullet, 7
 - definition, 8
 - directive table, 14
 - enumerated, 7
 - field, 9
 - horizontal, 7
 - itemize, 7
 - options, 9
- literal block, 9, 33
- literalinclude
 - directive, 34

M

- Makefile
 - sphinx, 45
- math expression, 41
- method
 - python directive, 34
- module
 - directive, 34
- myclass (*class in reST*), 11

N

- name
 - option, 16
- note
 - directive, 31

O

- only
 - directive, 31
- option
 - name, 16

P

- pngmath
 - extension, 41
- pull-quote
 - directive, 11
- python directive
 - attribute, 34
 - class, 34
 - classmethod, 34
 - currentmodule, 34
 - decorator, 34
 - exception, 34
 - method, 34
 - staticmethod, 34

Q

- quotes
 - block, 10

- line block, 10

R

- ref, 16
- reference, 16
 - citation style, 18
 - cross, 16
 - embedded, 18
 - in-line, 18
 - indirect, 16
 - internal, 16
 - label, 16
 - styled, 23
 - target, 16
- replace
 - directive, 23
- replacement, 23
 - image, 23
- reST (*module*), 1, 5
- role
 - cross-reference, 29
 - directive, 32
 - sphinx, 27
- rst-class
 - directive, 32
- rubric, 24

S

- see toc
 - table of contents, 30
 - toctree, 30
- seealso
 - directive, 31
- sidebar, 24
- simple
 - table, 12
- source code
 - directive, 34
 - include, 34
- sourcecode
 - directive, 23
- sphinx
 - autosectionlabel, 28
 - configuration, 45
 - cross-reference, 27
 - cross-reference document, 28
 - directive, 30
 - document cross-reference, 28
 - in-line markup, 27
 - Makefile, 45
 - role, 27
- Sphinx (*module*), 27
- sphinx-quickstart, 45
- standalone
 - hyperlink, 18
- staticmethod
 - python directive, 34
- strong, 6

substitution
 definition, 23

T

table, 12
 csv, directive, 14
 emacs, 14
 grid, 13
 latex, 15
 list, directive, 14
 of contents, 21
 simple, 12
tag, 31
target
 hypertext, 16
 reference, 16
title, 6
 hierarchy, 6
toc, 21
toctree
 directive, 30
topic, 24

U

unicode
 character code, 23
 directive, 23

W

warning
 directive, 31