
sbt-rtfd-markdown-test

Release 0.0.1a

Test

August 27, 2016

1	1	Misc rst Markup Examples	3
1.1	1.1	Giant tables	3
1.2	1.2	Sidebar	3
1.3	1.3	Boxes	4
1.4	1.4	Table: Every other row	4
1.5	1.5	Citation	5
1.6	1.6	Download links	5
1.7	1.7	Substitution	5
2	1	rst Structural Element Examples	7
2.1	1.1	Transitions	7
2.2	1.2	Inline Markup	8
2.3	1.3	Body Elements	8
2.4	1.4	Formatting	9
2.5	1.5	Literal Blocks	10
2.6	1.6	Tables	12
2.7	1.7	Footnotes	12
2.8	1.8	Citations	12
2.9	1.9	Targets	12
2.10	1.10	Directives	13
2.11	1.11	Images	13
2.12	1.12	Admonition Boxes	14
2.13	1.13	Topics, Sidebars, and Rubrics	15
2.14	1.14	Substitution	15
2.15	1.15	Comments	16
3	1	Examples of Code in rst	17
3.1	1.1	Showing Code Inline	17
3.2	1.2	Blocks of Code	18
3.3	1.3	Emphasized lines with line numbers	21
4		RST Header	23
4.1		2nd level rst header	23
5		External References are Footnotes	25
		Bibliography	27

These are some RST examples, using the Read The Docs template. This content is here just so I can do some noodling around with sbt, sphinx, github, and Read The Docs.



Fig. 1: **On the internet, no one knows you're a dog.**

Attention: This is version number comes from the <code>/src/sphinx/config.py</code> file: 0.0.1a

1 Misc rst Markup Examples

Table of Contents

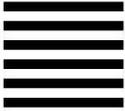
- 1 *Misc rst Markup Examples*
 - 1.1 *Giant tables*
 - 1.2 *Sidebar*
 - 1.3 *Boxes*
 - 1.4 *Table: Every other row*
 - 1.5 *Citation*
 - * 1.5.1 *Images*
 - 1.6 *Download links*
 - 1.7 *Substitution*

1.1 1.1 Giant tables

Header 1	Header 2	Header 3	Header 1	Header 2	Header 3	Header 1	Header 2	Header 3	Header 1	Header 2	Header 3
body row 1	col-umn 2	col-umn 3	body row 1	col-umn 2	col-umn 3	body row 1	col-umn 2	col-umn 3	body row 1	col-umn 2	col-umn 3
body row 1	col-umn 2	col-umn 3	body row 1	col-umn 2	col-umn 3	body row 1	col-umn 2	col-umn 3	body row 1	col-umn 2	col-umn 3
body row 1	col-umn 2	col-umn 3	body row 1	col-umn 2	col-umn 3	body row 1	col-umn 2	col-umn 3	body row 1	col-umn 2	col-umn 3
body row 1	col-umn 2	col-umn 3	body row 1	col-umn 2	col-umn 3	body row 1	col-umn 2	col-umn 3	body row 1	col-umn 2	col-umn 3

1.2 1.2 Sidebar

Ch'ien / The Creative



Above CH'IEN THE CREATIVE, HEAVEN

Below CH'IEN THE CREATIVE, HEAVEN

The first hexagram is made up of six unbroken lines. These unbroken lines stand for the primal power, which is light-giving, active, strong, and of the spirit. The hexagram is consistently strong in character, and since it is without weakness, its essence is power or energy. Its image is heaven. Its energy is represented as unrestricted by any fixed conditions in space and is therefore conceived of as motion. Time is regarded as the basis of this motion. Thus the hexagram includes also the power of time and the power of persisting in time, that is, duration.

The power represented by the hexagram is to be interpreted in a dual sense in terms of its action on the universe and of its action on the world of men. In relation to the universe, the hexagram expresses the strong, creative action of the Deity. In relation to the human world, it denotes the creative action of the holy man or sage, of the ruler or leader of men, who through his power awakens and develops their higher nature.

1.3 1.3 Boxes

Tip: Tip: Enable math extensions if you want equations to show up.

Note: This is a note about math equations.

Danger: Danger! Math can be addictive.

Warning: Warning: Math can be frustrating.

1.4 1.4 Table: Every other row

The default Read The Docs rst template formats every other line in a table with white text on a white background:

Example
Thing1
Thing2
Thing3

1.5 1.5 Citation

Here I am making a citation ¹, another ² and another ³

1.5.1 1.5.1 Images

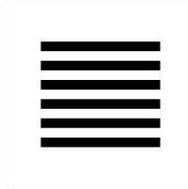


Fig. 1.1: This is a caption for a figure.

A figure is an image with a caption and/or a legend:



Fig. 1.2: Cats on the internet are fine. But dogs on the internet are **fanTAStic!**

1.6 1.6 Download links

This long download link should be blue with icon, and should wrap white-spaces

1.7 1.7 Substitution

See [substitutions in the official RST reference](#)

¹ This is the citation I made, let's make this extremely long so that we can tell that it doesn't follow the normal responsive table stuff.

² This citation has some `code` blocks in it, maybe some **bold** and *italics* too. Heck, lets put a link to a meta citation ³ too.

Original code for this page is from https://github.com/snide/sphinx_rtd_theme/blob/master/demo_docs/source/demo.rst

1 rst Structural Element Examples

Here's the quick reference for RST.

Table of Contents

- *1 rst Structural Element Examples*
 - *1.1 Transitions*
 - *1.2 Inline Markup*
 - *1.3 Body Elements*
 - * *1.3.1 Section Title*
 - * *1.3.2 Paragraphs*
 - * *1.3.3 Bullet Lists*
 - * *1.3.4 Enumerated Lists*
 - * *1.3.5 Definition Lists*
 - *1.4 Formatting*
 - * *1.4.1 Field Lists*
 - * *1.4.2 Option Lists*
 - *1.5 Literal Blocks*
 - * *1.5.1 Line Blocks*
 - * *1.5.2 Block Quotes*
 - * *1.5.3 Doctest Blocks*
 - *1.6 Tables*
 - *1.7 Footnotes*
 - *1.8 Citations*
 - *1.9 Targets*
 - * *1.9.1 External targets*
 - * *1.9.2 Target Footnotes*
 - *1.10 Directives*
 - *1.11 Images*
 - *1.12 Admonition Boxes*
 - *1.13 Topics, Sidebars, and Rubrics*
 - *1.14 Substitution*
 - *1.15 Comments*

2.1 1.1 Transitions

Here's a transition line:

It divides the section.

2.2 1.2 Inline Markup

Paragraphs contain text and may contain inline markup: *emphasis*, **strong emphasis**, inline literals, standalone hyperlinks (<http://www.python.org>), external hyperlinks ([Python](#)), internal cross-references (*example*), external hyperlinks with embedded URIs ([Python web site](#)), footnote references (manually numbered ¹, anonymous auto-numbered ³, labeled auto-numbered ², or symbolic ^{*0}), citation references (*[CIT2002]*), and inline hyperlink targets (see *Targets* below for a reference back to here). Character-level inline markup is also possible (although exceedingly ugly!) in *reStructuredText*.

The default role for interpreted text is *Title Reference*. Here are some explicit interpreted text roles: a PEP reference (**PEP 287**); an RFC reference (**RFC 2822**); a _{subscript}; a ^{superscript}; and explicit roles for *standard inline* markup.

Let's test wrapping and whitespace significance in inline literals: This is an example of --inline-literal --text, --including some-- strangely--hyphenated-words. Adjust-the-width-of-your-browser-window to see how the text is wrapped. -- ---- - Now note the spacing between the words of this sentence (words should be grouped in pairs).

If the --pep-references option was supplied, there should be a live link to PEP 258 here.

2.3 1.3 Body Elements

2.3.1 1.3.1 Section Title

That's a section title: the text just above this line.

2.3.2 1.3.2 Paragraphs

A paragraph.

2.3.3 1.3.3 Bullet Lists

- A bullet list
 - Nested bullet list.
 - Nested item 2.
- Item 2.
 - Paragraph 2 of item 2.
 - Nested bullet list.
 - Nested item 2.

¹ A footnote contains body elements, consistently indented by at least 3 spaces.

This is the footnote's second paragraph.

³ This footnote is numbered automatically and anonymously using a label of “#” only.

² Footnotes may be numbered, either manually (as in ¹) or automatically using a “#”-prefixed label. This footnote has a label so it can be referred to from multiple places, both as a footnote reference (²) and as a hyperlink reference (*label*).

⁰ Footnotes may also use symbols, specified with a “*” label. Here's a reference to the next footnote: †⁰.

- * Third level.
- * Item 2.
- Nested item 3.

2.3.4 1.3.4 Enumerated Lists

1. Arabic numerals.
 - (a) lower alpha
 - i. (lower roman)
 - A. upper alpha.
 - B. upper roman)
2. Lists that don't start at 1:
 - (a) Three
 - (b) Four
 - (a) C
 - (b) D
 - (a) iii
 - (b) iv
3. List items may also be auto-enumerated.

2.3.5 1.3.5 Definition Lists

Term Definition

Term [classifier] Definition paragraph 1.

Definition paragraph 2.

Term Definition

2.4 1.4 Formatting

Double-dashes – “–” – must be escaped somehow in HTML output.

2.4.1 1.4.1 Field Lists

what Field lists map field names to field bodies, like database records. They are often part of an extension syntax. They are an unambiguous variant of RFC 2822 fields.

how arg1 arg2 The field marker is a colon, the field name, and a colon.

The field body may contain one or more body elements, indented relative to the field marker.

Here's an example of a field list:

Field List

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

some text

Field List 2 Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor

2.4.2 1.4.2 Option Lists

For listing command-line options:

- a** command-line option “a”
- b file** options can have arguments and long descriptions
- long** options can be long also
- input=file** long options can also have arguments
- very-long-option** The description can also start on the next line.
The description may contain multiple body elements, regardless of where it starts.
- x, -y, -z** Multiple options are an “option group”.
- v, --verbose** Commonly-seen: short & long options.
- 1 file, --one=file, --two file** Multiple options with arguments.
- /V** DOS/VMS-style options too

There must be at least two spaces between the option and the description.

2.5 1.5 Literal Blocks

Literal blocks are indicated with a double-colon (“::”) at the end of the preceding paragraph (over there -->). They can be indented:

```
if literal_block:
    text = 'is left as-is'
    spaces_and_linebreaks = 'are preserved'
    markup_processing = None
```

Or they can be quoted without indentation:

```
>> Great idea!
>
> Why didn't I think of that?
```

2.5.1 1.5.1 Line Blocks

This is a line block. It ends with a blank line.

Each new line begins with a vertical bar (“|”).

Line breaks and initial indents are preserved.

Continuation lines are wrapped portions of long lines; they begin with a space in place of the vertical bar.

The left edge of a continuation line need not be aligned with the left edge of the text above it.

This is a second line block.

Blank lines are permitted internally, but they must begin with a “|”.

Take it away, Eric the Orchestra Leader!

A one, two, a one two three four

Half a bee, philosophically,
must, *ipso facto*, half not be.

But half the bee has got to be,
vis a vis its entity. D’you see?

But can a bee be said to be
or not to be an entire bee,
when half the bee is not a bee,
due to some ancient injury?

Singing...

2.5.2 1.5.2 Block Quotes

Block quotes consist of indented body elements:

My theory by A. Elk. Brackets Miss, brackets. This theory goes as follows and begins now. All brontosaurus are thin at one end, much much thicker in the middle and then thin again at the far end. That is my theory, it is mine, and belongs to me and I own it, and what it is too.

—Anne Elk (Miss)

2.5.3 1.5.3 Doctest Blocks

```
>>> print 'Python-specific usage examples; begun with ">>>"
Python-specific usage examples; begun with ">>>"
>>> print '(cut and pasted from interactive Python sessions)'
(cut and pasted from interactive Python sessions)
```

2.6 1.6 Tables

Here's a grid table followed by a simple table:

Header row, column 1 (header rows optional)	Header 2	Header 3	Header 4
body row 1, column 1	column 2	column 3	column 4
body row 2	Cells may span columns.		
body row 3	Cells may span rows.		
body row 4			
body row 5	Cells may also be empty: -->	Table cells	

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

contain

body elements.

2.7 1.7 Footnotes

2.8 1.8 Citations

Here's a reference to the above, [\[CIT2002\]](#)

2.9 1.9 Targets

This paragraph is pointed to by the explicit `_example` target. A reference can be found under *Inline Markup*, above. *Inline hyperlink targets* are also possible.

Section headers are implicit targets, referred to by name. See *Targets*, which is a subsection of *Body Elements*.

2.9.1 1.9.1 External targets

Explicit external targets are interpolated into references such as “[Python](#)”.

Here's a reference to the Definitive [RST Reference](#) documentation.

You can refer to another rst document within the site with a **Sphinx** directive. A reference to the *1 Examples of Code in rst* like this: `:ref: `rst_code``

Targets may be indirect and anonymous. Thus *this phrase* may also refer to the *Targets* section.

2.9.2 1.9.2 Target Footnotes

If you use the `.. target-notes::` directive, footnotes for **all external references** will be generated, and the footnotes themselves will be put after that directive. (Thus you usually want to put the directive at the bottom of a document so the footnotes will be at the bottom – the *foot* of the document.)

Target footnoes are not used in this document. But you can see it in action in *this one* `<rst_tiny>`.

2.10 1.10 Directives

These are just a sample of the many reStructuredText Directives. For others, please see <http://docutils.sourceforge.net/docs/ref/rst/directives.html>.

An example of the “contents” directive can be seen above this section (a local, untitled table of *contents*) and at the beginning of the document (a document-wide *table of contents*).

2.11 1.11 Images

An image directive with a link (target) to the Targets section. (The image is a clickable link):



A figure is an image with a caption and/or a legend:



Fig. 2.1: Cats on the internet are fine. But dogs on the internet are **fanTAStic!**

A figure directive with center alignment and width of 100. (If you click on it, you'll see the lovely full-sized image.)



2.12 1.12 Admonition Boxes

Attention: Attention - Directives at large.

Caution: Don't take any wooden nickels.

Danger: Mad scientist at work!

Error: Does not compute.

Hint: It's bigger than a bread box.

Important: These things are important: - Wash behind your ears. - Be nice. - Clean up your room. - Back up your data.

Note: This is a note.

Tip: 15% if the service is good.

Warning: Strong prose may provoke extreme mental exertion. Reader discretion is strongly advised.

And, by the way...

You can make up your own admonition too.

2.13 1.13 Topics, Sidebars, and Rubrics

Sidebar Title

This a subtitle

This is a sidebar. It is for text outside the flow of the main text. The subtitle above doesn't have any interesting formatting so it's hard to tell that it has any import at all.

This is a rubric inside a sidebar

Sidebars often appears beside the main text with a border and background color.

Topic Title

This is a topic.

This is a rubric

This paragraph contains a literal block:

```
Connecting... OK
Transmitting data... OK
Disconnecting... OK
```

and thus consists of a simple paragraph, a literal block, and another simple paragraph. Nonetheless it is semantically *one* paragraph.

This construct is called a *compound paragraph* and can be produced with the “compound” directive.

2.14 1.14 Substitution

An inline image example: Instead of showing the words `biohazard`, show ()

The code to accomplish a substitution (a.k.a. *replacement*) is:

```
An inline image example:  Instead of showing the words ``biohazard``, show (|biohazard|)
.. |biohazard| image:: static/tiny-Biohazard_symbol.png
```

I recommend that you try [Smalltalk](#), *the* best language around.

In the preceding text, `|`Python web site <http://www.python.org>`__|` was replaced with ``Smalltalk <http://c2.com/cgi/wiki?SmalltalkLanguage>`__`

2.15 1.15 Comments

Here's one:

Of course you can't see it, because it's a comment in the source for this file. Here's the what the rst for the comment looks like in the rst source for this file:

```
.. Comments begin with two dots and a space. Anything may  
follow, except for the syntax of footnotes, hyperlink  
targets, directives, or substitution definitions.
```

1 Examples of Code in rst

Table of Contents

- 1 Examples of Code in rst
 - 1.1 Showing Code Inline
 - * 1.1.1 Inline code and references
 - 1.2 Blocks of Code
 - * 1.2.1 class directive and parameter args
 - * 1.2.2 parsed-literal
 - * 1.2.3 code-block
 - * 1.2.4 Include code from a source file
 - * 1.2.5 literalinclude includes content from a file without interpreting it
 - * 1.2.6 parsed-literal shows text without interpreting it
 - 1.3 Emphasized lines with line numbers

3.1 1.1 Showing Code Inline

Error: TBD

3.1.1 1.1.1 Inline code and references

`reStructuredText` is a markup language. It can use roles and declarations to turn reST into HTML.

In reST, `*hello world*` becomes `hello world`. This is because a library called `Docutils` was able to parse the reST and use a `Writer` to output it that way.

If I type ```an inline literal``` it will wrap it in `<tt>`. You can see more details on the [Inline Markup](#) on the `Docutils` homepage.

Also with `sphinx.ext.autodoc`, which I use in the demo, I can link to `test_py_module.test.Foo`. It will link you right to my code documentation for it.

3.2 1.2 Blocks of Code

3.2.1 1.2.1 class directive and parameter args

At this point optional parameters *cannot be generated from code*. However, some projects will manually do it, like so:

This example comes from [django-payments module docs](#).

```
class payments.dotpay.DotpayProvider (seller_id, pin[, channel=0[, lock=False ], lang='pl' ])
```

param seller_id Seller ID assigned by Dotpay

param pin PIN assigned by Dotpay

param channel Default payment channel (consult reference guide)

param lang UI language

param lock Whether to disable channels other than the default selected above

This backend implements payments using a popular Polish gateway, [Dotpay.pl](#).

Due to API limitations there is no support for transferring purchased items.

This example uses the `.. class::` directive to format `payments.dotpay.DotpayProvider(seller_id, pin[, channel=0[, lock=False], lang='pl'])` and automatically put the word *class* in front of it. The `:param` args follow the class line. The optional `:param` args nicely format the parameters.

Here's the rst for the entire example above, including the references:

At this point optional parameters *cannot be generated from code*.

However, some projects will manually do it, like so:

This example comes from [django-payments module docs](#).

```
.. class:: payments.dotpay.DotpayProvider(seller_id, pin[, channel=0[, lock=False], lang='pl'] )
```

```
:param seller_id: Seller ID assigned by Dotpay
```

```
:param pin: PIN assigned by Dotpay
```

```
:param channel: Default payment channel (consult reference guide)
```

```
:param lang: UI language
```

```
:param lock: Whether to disable channels other than the default selected above
```

This backend implements payments using a popular Polish gateway, [Dotpay.pl](#).

Due to API limitations there is no support for transferring purchased items.

```
.. _cannot be generated from code: https://groups.google.com/forum/#!topic/sphinx-users/\_q
```

```
.. _django-payments module docs: http://django-payments.readthedocs.org/en/latest/modules.html
```

3.2.2 1.2.2 parsed-literal

Using the `.. parsed-literal::` directive will show text without reformatting it. It will do some simple formatting (coloring) but will often show the text in an ugly, large monospaced font. (Although you can change this with your own rst template.)

```
# parsed-literal test  
curl -O http://someurl/release-0.0.1a.tar-gz
```

Here's the rst for the above:

```
.. parsed-literal::

    # parsed-literal test
    curl -O http://someurl/release-0.0.1a.tar-gz
```

3.2.3 1.2.3 code-block

The `.. code-block::` directive looks much better than `.. parsed-literal::`. (At least with the default Read The Docs template.) The `.. code-block::` directive will try to display the text that follows it as programming code. If you specify a language that rst knows about, it will also format the text in a way that makes sense for that language.

Here's an example of a code block without any formatting or coloring:

```
print "Hello world"

def some_function():
    interesting = False
    if interesting print 'This is a nonsensical function.'
```

The language specified is *text*. That tells rst to **not** apply any formatting or coloring.

```
text
.. code-block:: text

    print "Hello world"

    def some_function():
        interesting = False
        if interesting print 'This is a nonsensical function.'
```

You **must** specify a language after `.. code-block::`. If you don't rst will produce a console warning and won't display the code at all.

Here's that same example with *ruby* specified as the language. With the language specified, keywords are now nicely colored:

```
print "Hello world"
def some_function():
    interesting = False
    if interesting print 'This is a nonsensical function.'
```

Here's the rst; the only difference is the word *ruby* after the `.. code-block::` directive to specify the language. (Note the space required between `.. code-block::` and *ruby*!)

```
.. code-block:: ruby

    print "Hello world"

    def some_function():
        interesting = False
        if interesting print 'This is a nonsensical function.'
```

```

{
  "windows": [
    {
      "panes": [
        {
          "shell_command": [
            "echo 'did you know'",
            "echo 'you can inline'"
          ]
        },
        {
          "shell_command": "echo 'single commands'"
        },
        "echo 'for panes'"
      ],
      "window_name": "long form"
    }
  ],
  "session_name": "shorthands"
}

```

3.2.4 1.2.4 Include code from a source file

Listing 3.1: Live code from /src/main/scala/common/package.scala

```

1 import java.util.concurrent._
2 import scala.util.DynamicVariable
3
4 package object common {
5
6   val forkJoinPool = new ForkJoinPool
7
8   abstract class TaskScheduler {
9     def schedule[T](body: => T): ForkJoinTask[T]
10    def parallel[A, B](taskA: => A, taskB: => B): (A, B) = {
11      val right = task {
12        taskB
13      }
14      val left = taskA
15      (left, right.join())
16    }
17  }
18
19  class DefaultTaskScheduler extends TaskScheduler {
20    def schedule[T](body: => T): ForkJoinTask[T] = {
21      val t = new RecursiveTask[T] {
22        def compute = body
23      }
24      Thread.currentThread match {
25        case wt: ForkJoinWorkerThread =>
26          t.fork()
27        case _ =>
28          forkJoinPool.execute(t)
29      }
30      t

```

```

31     }
32   }
33
34   val scheduler =
35     new DynamicVariable[TaskScheduler] (new DefaultTaskScheduler)
36
37   def task[T] (body: => T): ForkJoinTask[T] = {
38     scheduler.value.schedule (body)
39   }

```

This is taken from the actual source file. The rst code can reference either absolute file paths or relative file paths. (Absolute file paths are brittle, so should be avoided of course.) With *relative file paths*, rst can only refer to files under the project's `.../src/sphinx` directory. But you can create a symbolic link in `.../src/sphinx` that refers to some other file. That's how this code is referenced: within rst, it refers to a file in `.../src/sphinx` that is actually a symbolic link to the real source file.

3.2.5 1.2.5 literalinclude includes content from a file without interpreting it

If you want to include the contents of a file but not have rst interpret it, use the *literalinclude* directive. For example, if you want to show the contents of a file (or parts of it) but that file contains text that are rst directives or some other code that rst might read as instructions and you don't want rst to execute (do) those directives, use *literalinclude*.

(TBD: link to the rst documentation)

Here's how *literalinclude* was used to read lines from a source file and include them above:

```

.. literalinclude:: /package.scala
   :language: scala
   :linenos:
   :lines: 1-40
   :caption: Live code from /src/main/scala/common/package.scala

```

3.2.6 1.2.6 parsed-literal shows text without interpreting it

And here is how to get the above to show as code and not be interpreted by the rst parser as directives:

```

.. parsed-literal::
   .. literalinclude:: /package.scala
      :language: scala
      :linenos:
      :lines: 1-40
      :caption: Live code from /src/main/scala/common/package.scala

```

3.3 1.3 Emphasized lines with line numbers

```

1   def some_function():
2     interesting = False
3     print 'This line is highlighted.'
4     print 'This one is not...'
5     print '...but this one is.'

```

You can use the `:linenos` and `:emphasize-lines:` codes to show line numbers and to highlight specific lines of code, respectively.

Here's the rst used for the above ruby code:

```
.. code-block:: ruby
   :linenos:
   :emphasize-lines: 3,5

   def some_function():
       interesting = False
       print 'This line is highlighted.'
       print 'This one is not...'
       print '...but this one is.'
```

RST Header

Note: This is awesome

4.1 2nd level rst header

More content

```
print "Hello world"
```

External References are Footnotes

This document uses the `.. target-notes::` directive, so all external references are shown as footnotes. Here's a reference to the Definitive [RST](http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html)¹ Reference documentation. And here's the [quick reference for RST](http://docutils.sourceforge.net/docs/user/rst/quickref.html)².

¹ <http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html>

² <http://docutils.sourceforge.net/docs/user/rst/quickref.html>

Bibliography

[CIT2002] Citations are text-labeled footnotes. They may be rendered separately and differently from footnotes.

P

payments.dotpay.DotpayProvider (built-in class), 18
Python Enhancement Proposals
 PEP 287, 8

R

RFC
 RFC 2822, 8