Quick *re*Structured*Text*

http://docutils.sourceforge.net/docs/user/rst/quickref.html
Being a cheat-sheet for reStructuredText
Updated $Date: 2009-07-14 14:05:34 +0000 (Tue, 14 Jul 2009) $

The full details of the markup may be found on the reStructuredText page. This document is just intended as a reminder.

Links that look like "(details)" point into the HTML version of the full reStructuredText specification document. These are relative links; if they don't work, please use the master "Quick reStructuredText" document.

Contents

Inline Markup

(details)

Inline markup allows words and phrases within text to have character styles (like italics and boldface) and functionality (like hyperlinks).

Plain text	Typical result	Notes
emphasis	*emphasis*	Normally rendered as italics.
strong emphasis	**strong emphasis**	Normally rendered as boldface.
`interpreted text`	(see note at right)	The rendering and *meaning* of interpreted text is domain- or application-dependent. It can be used for things like index entries or explicit descriptive markup (like program identifiers).
``inline literal``	inline literal	Normally rendered as monospaced text. Spaces should be preserved, but line breaks will not be.
reference_	reference	A simple, one-word hyperlink reference. See Hyperlink Targets.
`phrase reference`_	phrase reference	A hyperlink reference with spaces or punctuation needs to be quoted with backquotes. See Hyperlink Targets.
anonymous__	anonymous	With two underscores instead of one, both simple and phrase references may be anonymous (the reference text is not repeated at the target). See Hyperlink Targets.
_`inline internal target`	inline internal target	A crossreference target within text. See Hyperlink

Plain text	Typical result	Notes
		Targets.
`|substitution reference|`	(see note at right)	The result is substituted in from the substitution definition. It could be text, an image, a hyperlink, or a combination of these and others.
`footnote reference [1]_`	footnote reference [1]	See Footnotes.
`citation reference [CIT2002]_`	citation reference [CIT2002]	See Citations.
`http://docutils.sf.net/`	http://docutils.sf.net/	A standalone hyperlink.

Asterisk, backquote, vertical bar, and underscore are inline delimiter characters. Asterisk, backquote, and vertical bar act like quote marks; matching characters surround the marked-up word or phrase, whitespace or other quoting is required outside them, and there can't be whitespace just inside them. If you want to use inline delimiter characters literally, escape (with backslash) or quote them (with double backquotes; i.e. use inline literals).

In detail, the reStructuredText specification says that in inline markup, the following rules apply to start-strings and end-strings (inline markup delimiters):

1. The start-string must start a text block or be immediately preceded by whitespace or any of `' " ( [ {` or `<`.
2. The start-string must be immediately followed by non-whitespace.
3. The end-string must be immediately preceded by non-whitespace.
4. The end-string must end a text block (end of document or followed by a blank line) or be immediately followed by whitespace or any of `' " . , : ; ! ? - ) ] } / \` or `>`.
5. If a start-string is immediately preceded by one of `' " ( [ {` or `<`, it must not be immediately followed by the corresponding character from `' " ) ] }` or `>`.
6. An end-string must be separated by at least one character from the start-string.
7. An unescaped backslash preceding a start-string or end-string will disable markup recognition, except for the end-string of inline literals.

Also remember that inline markup may not be nested (well, except that inline literals can contain any of the other inline markup delimiter characters, but that doesn't count because nothing is processed).

Escaping with Backslashes

(details)

reStructuredText uses backslashes ("\") to override the special meaning given to markup characters and get the literal characters themselves. To get a literal backslash, use an escaped backslash ("\\"). For example:

Raw reStructuredText	**Typical result**
escape ``with`` "\"	*escape* with ""
escape \``with`` "\\"	*escape* ``with`` "\"

In Python strings it will, of course, be necessary to escape any backslash characters so that they actually *reach* reStructuredText. The simplest way to do this is to use raw strings:

Python string	**Typical result**
r"""*escape* \`with` "\\""""	*escape* `with` "\"
"""\\*escape* \\`with` "\\\\""""	*escape* `with` "\"
"""*escape* \`with` "\\""""	*escape* with ""

Section Structure

(details)

Plain text	**Typical result**
===== Title ===== Subtitle -------- Titles are underlined (or over- and underlined) with a printing nonalphanumeric 7-bit ASCII character. Recommended choices are "``= - ` : ' " ~ ^ _ * + # < >``". The underline/overline must be at least as long as the title text. A lone top-level (sub)section is lifted up to be the document's (sub)title.	**Title** **Subtitle** Titles are underlined (or over- and underlined) with a printing nonalphanumeric 7-bit ASCII character. Recommended choices are "= - ` : ' " ~ ^ _ * + # < >". The underline/overline must be at least as long as the title text. A lone top-level (sub)section is lifted up to be the document's (sub)title.

Paragraphs

(details)

Plain text	Typical result

`This is a paragraph.`
 	
`Paragraphs line up at their left`
`edges, and are normally separated`
`by blank lines.`
	This is a paragraph.
	
	Paragraphs line up at their left edges, and are normally separated by blank lines.

Bullet Lists

(details)

Plain text	Typical result

`Bullet lists:`
 	
`- This is item 1`
`- This is item 2`
 	
`- Bullets are "-", "*" or "+".`
`  Continuing text must be aligned`
`  after the bullet and whitespace.`
 	
`Note that a blank line is required`
`before the first item and after the`
`last, but is optional between items.`
	Bullet lists:
	
	• This is item 1
	• This is item 2
	• Bullets are "-", "*" or "+". Continuing text must be aligned after the bullet and whitespace.
	
	Note that a blank line is required before the first item and after the last, but is optional between items.

Enumerated Lists

(details)

Plain text	Typical result

`Enumerated lists:`
 	
`3. This is the first item`
`4. This is the second item`
`5. Enumerators are arabic numbers,`
`   single letters, or roman numerals`
`6. List items should be sequentially`
	Enumerated lists:
	
	3. This is the first item
	4. This is the second item
	5. Enumerators are arabic numbers, single letters, or roman

Plain text	Typical result
numbered, but need not start at 1 (although not all formatters will honour the first index). #. This item is auto-enumerated	numerals 6. List items should be sequentially numbered, but need not start at 1 (although not all formatters will honour the first index). 7. This item is auto-enumerated

Definition Lists

(details)

Plain text	Typical result
Definition lists: what Definition lists associate a term with a definition. how The term is a one-line phrase, and the definition is one or more paragraphs or body elements, indented relative to the term. Blank lines are not allowed between term and definition.	Definition lists: **what** Definition lists associate a term with a definition. **how** The term is a one-line phrase, and the definition is one or more paragraphs or body elements, indented relative to the term. Blank lines are not allowed between term and definition.

Field Lists

(details)

Plain text	Typical result
:Authors: Tony J. (Tibs) Ibbs, David Goodger (and sundry other good-natured folks) :Version: 1.0 of 2001/08/08 :Dedication: To my father.	**Authors:** Tony J. (Tibs) Ibbs, David Goodger (and sundry other good-natured folks) **Version:** 1.0 of 2001/08/08 **Dedication:** To my father.

Field lists are used as part of an extension syntax, such as options for directives, or database-like records meant for further processing. Field lists may also be used as generic two-column table constructs in documents.

Option Lists

(details)

Plain text	Typical result

`-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`
`/V            DOS/VMS-style options too`
	`-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
	`/V` DOS/VMS-style options too

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

Literal Blocks

(details)

Plain text	Typical result

`A paragraph containing only two colons`
`indicates that the following indented`
`or quoted text is a literal block.`
 	
`::`
 	
`  Whitespace, newlines, blank lines, and`
`  all kinds of markup (like *this* or`
`  \this) is preserved by literal blocks.`
 	
`  The paragraph containing only '::'`
`  will be omitted from the result.`
 	
``The ``::`` may be tacked onto the very``
``end of any paragraph. The ``::`` will be``
`omitted if it is preceded by whitespace.`
``The ``::`` will be converted to a single``
`colon if preceded by text, like this::`
 	
`  It's very convenient to use this form.`
	A paragraph containing only two colons indicates that the following indented or quoted text is a literal block.
	
	`Whitespace, newlines, blank lines, and`
	`all kinds of markup (like *this* or`
	`\this) is preserved by literal blocks.`
	
	`The paragraph containing only '::'`
	`will be omitted from the result.`
	
	The `::` may be tacked onto the very end of any paragraph. The `::` will be omitted if it is preceded by whitespace. The `::` will be converted to a single colon if preceded by text, like this:

Plain text	Typical result

`Literal blocks end when text returns to`
`the preceding paragraph's indentation.`
`This means that something like this`
`is possible::`
 	
`      We start here`
`    and continue here`
`  and end here.`
 	
`Per-line quoting can also be used on`
`unindented literal blocks::`
 	
`> Useful for quotes from email and`
`> for Haskell literate programming.`
	`It's very convenient to use this form.`
	
	Literal blocks end when text returns to the preceding paragraph's indentation. This means that something like this is possible:
	
	`    We start here`
	`  and continue here`
	`and end here.`
	
	Per-line quoting can also be used on unindented literal blocks:
	
	`> Useful for quotes from email and`
	`> for Haskell literate programming.`

Line Blocks

(details)

Plain text	Typical result

`| Line blocks are useful for addresses,`
`| verse, and adornment-free lists.`
`|`
`| 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 spaces in place of vertical bars.`
	Line blocks are useful for addresses,
	verse, and adornment-free lists.
	
	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 spaces in place of vertical bars.

Block Quotes

(details)

Plain text	Typical result

`Block quotes are just:`
 	
`    Indented paragraphs,`
	Block quotes are just:
	
	    Indented paragraphs,

<table>
<tr><th>Plain text</th><th>Typical result</th></tr>
<tr><td><pre>
        and they may nest.
</pre></td><td>and they may nest.</td></tr>
</table>

Use empty comments to separate indentation contexts, such as block quotes and directive contents.

Doctest Blocks

(details)

<table>
<tr><th>Plain text</th><th>Typical result</th></tr>
<tr><td><pre>
Doctest blocks are interactive
Python sessions. They begin with
"``>>>``" and end with a blank line.

>>> print "This is a doctest block."
This is a doctest block.
</pre></td><td>Doctest blocks are interactive Python sessions. They begin with ">>>" and end with a blank line.
<pre>
>>> print "This is a doctest block."
This is a doctest block.
</pre></td></tr>
</table>

"The doctest module searches a module's docstrings for text that looks like an interactive Python session, then executes all such sessions to verify they still work exactly as shown." (From the doctest docs.)

Tables

(details)

There are two syntaxes for tables in reStructuredText. Grid tables are complete but cumbersome to create. Simple tables are easy to create but limited (no row spans, etc.).

<table>
<tr><th>Plain text</th><th>Typical result</th></tr>
<tr><td><pre>
Grid table:

+------------+------------+-----------+
| Header 1 | Header 2 | Header 3 |
+============+============+===========+
| body row 1 | column 2 | column 3 |
+------------+------------+-----------+
| body row 2 | Cells may span columns.|
+------------+------------+-----------+
</pre></td><td>Grid table:
<table>
<tr><th>Header 1</th><th>Header 2</th><th>Header 3</th></tr>
<tr><td>body row 1</td><td>column 2</td><td>column 3</td></tr>
<tr><td>body row 2</td><td colspan="2">Cells may span columns.</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Plain text</th><th>Typical result</th></tr>
<tr><td><pre>
| body row 3 | Cells may | - Cells |
+------------+ span rows. | - contain |
| body row 4 | | - blocks. |
+------------+------------+-----------+
</pre></td><td>
<table>
<tr><th>Header 1</th><th>Header 2</th><th>Header 3</th></tr>
<tr><td>body row 3</td><td rowspan="2">Cells may span rows.</td><td rowspan="2">• Cells
• contain
• blocks.</td></tr>
<tr><td>body row 4</td></tr>
</table>
</td></tr>
<tr><td><pre>
Simple table:

===== ===== ======
   Inputs   Output
------------ ------
  A   B   A or B
===== ===== ======
False False False
True   False True
False True   True
True   True   True
===== ===== ======
</pre></td><td>
Simple table:
<table>
<tr><th colspan="2">Inputs</th><th>Output</th></tr>
<tr><th>A</th><th>B</th><th>A or B</th></tr>
<tr><td>False</td><td>False</td><td>False</td></tr>
<tr><td>True</td><td>False</td><td>True</td></tr>
<tr><td>False</td><td>True</td><td>True</td></tr>
<tr><td>True</td><td>True</td><td>True</td></tr>
</table>
</td></tr>
</table>

Transitions

(details)

<table>
<tr><th>Plain text</th><th>Typical result</th></tr>
<tr><td><pre>
A transition marker is a horizontal line
of 4 or more repeated punctuation
characters.

A transition should not begin or end a
section or document, nor should two
transitions be immediately adjacent.
</pre></td><td>
A transition marker is a horizontal line of 4 or more repeated punctuation characters.

A transition should not begin or end a section or document, nor should two transitions be immediately adjacent.
</td></tr>
</table>

Transitions are commonly seen in novels and short fiction, as a gap spanning one or more lines, marking text divisions or signaling changes in subject, time, point of view, or emphasis.

Explicit Markup

Explicit markup blocks are used for constructs which float (footnotes), have no direct paper-document representation (hyperlink targets, comments), or require specialized processing (directives). They all begin with two periods and whitespace, the "explicit markup start".

Footnotes

(details)

Plain text	Typical result

`Footnote references, like [5]_.`
`Note that footnotes may get`
`rearranged, e.g., to the bottom of`
`the "page".`
 	
`.. [5] A numerical footnote. Note`
`   there's no colon after the ``]``.`
	Footnote references, like [5]. Note that footnotes may get rearranged, e.g., to the bottom of the "page".
	
	[5] A numerical footnote. Note there's no colon after the `]`.

`Autonumbered footnotes are`
`possible, like using [#]_ and [#]_.`
 	
`.. [#] This is the first one.`
`.. [#] This is the second one.`
 	
`They may be assigned 'autonumber`
`labels' - for instance,`
`[#fourth]_ and [#third]_.`
 	
`.. [#third] a.k.a. third_`
 	
`.. [#fourth] a.k.a. fourth_`
	Autonumbered footnotes are possible, like using [1] and [2].
	
	They may be assigned 'autonumber labels' - for instance, [4] and [3].
	
	[1] This is the first one.
	[2] This is the second one.
	[3] a.k.a. third
	[4] a.k.a. fourth

`Auto-symbol footnotes are also`
`possible, like this: [*]_ and [*]_.`
 	
`.. [*] This is the first one.`
`.. [*] This is the second one.`
	Auto-symbol footnotes are also possible, like this: [*] and [†].
	
	[*] This is the first symbol footnote
	[†] This is the second one.

The numbering of auto-numbered footnotes is determined by the order of the footnotes, not of the references. For auto-numbered footnote references without autonumber labels ("`[#]_`"), the references and footnotes must be in the same relative order. Similarly for auto-symbol footnotes ("`[*]_`").

Citations

(details)

Plain text	Typical result
Citation references, like [CIT2002]_. Note that citations may get rearranged, e.g., to the bottom of the "page". .. [CIT2002] A citation (as often used in journals). Citation labels contain alphanumerics, underlines, hyphens and fullstops. Case is not significant. Given a citation like [this]_, one can also refer to it like this_. .. [this] here.	Citation references, like [CIT2002]. Note that citations may get rearranged, e.g., to the bottom of the "page". Citation labels contain alphanumerics, underlines, hyphens and fullstops. Case is not significant. Given a citation like [this], one can also refer to it like this. **[CIT2002]** A citation (as often used in journals). **[this]** here.

Hyperlink Targets

(details)

External Hyperlink Targets

Plain text	Typical result
External hyperlinks, like Python_. .. _Python: http://www.python.org/	*Fold-in form* External hyperlinks, like Python.
	Call-out form External hyperlinks, like *Python*. *Python:* http://www.python.org/

"*Fold-in*" is the representation typically used in HTML documents (think of the indirect hyperlink being "folded in" like ingredients into a cake), and "*call-out*" is more suitable for printed documents, where the link needs to be presented explicitly, for example as a footnote. You can force usage of the call-out form by using the "target-notes" directive.

reStructuredText also provides for **embedded URIs** (details), a convenience at the expense of readability. A hyperlink reference may directly embed a target URI inline, within angle brackets. The following is exactly equivalent to the example above:

Plain text	Typical result
External hyperlinks, like `Python <http://www.python.org/>`_.	External hyperlinks, like Python.

Internal Hyperlink Targets

Plain text	Typical result
Internal crossreferences, like example_. .. _example: This is an example crossreference target.	*Fold-in form* Internal crossreferences, like example This is an example crossreference target.
	Call-out form Internal crossreferences, like example *example:* This is an example crossreference target.

Indirect Hyperlink Targets

(details)

Plain text	Typical result
Python_ is `my favourite programming language`__. .. _Python: http://www.python.org/ __ Python_	Python is my favourite programming language.

The second hyperlink target (the line beginning with "__") is both an indirect hyperlink target (*indirectly* pointing at the Python website via the "`Python_`" reference) and an **anonymous hyperlink target**. In the text, a double-underscore suffix is used to indicate an **anonymous hyperlink reference**. In an anonymous hyperlink target, the reference text is not repeated. This is useful for references with long text or throw-away references, but the target should be kept close to the reference to prevent them going out of sync.

Implicit Hyperlink Targets

(details)

Section titles, footnotes, and citations automatically generate hyperlink targets (the title text or footnote/citation label is used as the hyperlink name).

Plain text	Typical result
Titles are targets, too ======================= Implict references, like `Titles are targets, too`_.	**Titles are targets, too** Implict references, like Titles are targets, too.

Directives

(details)

Directives are a general-purpose extension mechanism, a way of adding support for new constructs without adding new syntax. For a description of all standard directives, see reStructuredText Directives.

Plain text	Typical result
For instance: .. image:: images/ball1.gif	For instance:

Substitution References and Definitions

(details)

Substitutions are like inline directives, allowing graphics and arbitrary constructs within text.

Plain text	Typical result

<table>
<tr><th>Plain text</th><th>Typical result</th></tr>
<tr><td>The |biohazard| symbol must be
used on containers used to
dispose of medical waste.

.. |biohazard| image:: biohazard.png</td><td>The ☣ symbol must be used on containers used to dispose of medical waste.</td></tr>
</table>

Comments

(details)

Any text which begins with an explicit markup start but doesn't use the syntax of any of the constructs above, is a comment.

<table>
<tr><th>Plain text</th><th>Typical result</th></tr>
<tr><td>.. This text will not be shown
   (but, for instance, in HTML might be
   rendered as an HTML comment)</td><td></td></tr>
<tr><td>An "empty comment" does not
consume following blocks.
(An empty comment is ".." with
blank lines before and after.)

..

        So this block is not "lost",
        despite its indentation.</td><td>An "empty comment" does not consume following blocks. (An empty comment is ".." with blank lines before and after.)

So this block is not "lost", despite its indentation.</td></tr>
</table>

Getting Help

Users who have questions or need assistance with Docutils or reStructuredText should post a message to the Docutils-Users mailing list. The Docutils project web site has more information.

Authors: Tibs (`tibs@tibsnjoan.co.uk`) and David Goodger (goodger@python.org)