reStructuredText Primer

This section is a brief introduction to reStructuredText (reST) concepts and syntax, intended to provide authors with enough information to author documents productively. Since reST was designed to be a simple, unobtrusive markup language, this will not take too long.

See also

The authoritative reStructuredText User Documentation. The "ref" links in this document link to the description of the individual constructs in the reST reference.

Paragraphs

The paragraph (ref) is the most basic block in a reST document. Paragraphs are simply chunks of text separated by one or more blank lines. As in Python, indentation is significant in reST, so all lines of the same paragraph must be left-aligned to the same level of indentation.

Inline markup

The standard reST inline markup is quite simple: use

- one asterisk: `*text*` for emphasis (italics),
- two asterisks: `**text**` for strong emphasis (boldface), and
- backquotes: ``` ``text`` ``` for code samples.

If asterisks or backquotes appear in running text and could be confused with inline markup delimiters, they have to be escaped with a backslash.

Be aware of some restrictions of this markup:

- it may not be nested,
- content may not start or end with whitespace: `* text*` is wrong,
- it must be separated from surrounding text by non-word characters. Use a backslash escaped space to work around that: `thisis\ *one*\ word`.

These restrictions may be lifted in future versions of the docutils.

reST also allows for custom "interpreted text roles", which signify that the enclosed text should be interpreted in a specific way. Sphinx uses this to provide semantic markup and cross-referencing of identifiers, as described in the appropriate section. The general syntax is `:rolename:`content``.

Standard reST provides the following roles:

- emphasis – alternate spelling for `*emphasis*`
- strong – alternate spelling for `**strong**`
- literal – alternate spelling for ``` ``literal`` ```
- subscript – subscript text
- superscript – superscript text
- title-reference – for titles of books, periodicals, and other materials

See Inline markup for roles added by Sphinx.

Lists and Quote-like blocks

List markup (ref) is natural: just place an asterisk at the start of a paragraph and indent properly. The same goes for numbered lists; they can also be autonumbered using a # sign:

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

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

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

Nested lists are possible, but be aware that they must be separated from the parent list items by blank lines:

```
* this is
* a list

  * with a nested list
  * and some subitems

* and here the parent list continues
```

Definition lists (ref) are created as follows:

```
term (up to a line of text)
   Definition of the term, which must be indented

   and can even consist of multiple paragraphs

next term
   Description.
```

Note that the term cannot have more than one line of text.

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

Line blocks (ref) are a way of preserving line breaks:

```
| These lines are
| broken exactly like in
| the source file.
```

There are also several more special blocks available:

- field lists (ref)
- option lists (ref)
- quoted literal blocks (ref)
- doctest blocks (ref)

Source Code

Literal code blocks (ref) are introduced by ending a paragraph with the special marker `::`. The literal block must be indented (and, like all paragraphs, separated from the surrounding ones by blank lines):

```
This is a normal text paragraph. The next paragraph is a code sample::

   It is not processed in any way, except
   that the indentation is removed.

   It can span multiple lines.

This is a normal text paragraph again.
```

The handling of the `::` marker is smart:

- If it occurs as a paragraph of its own, that paragraph is completely left out of the document.

- If it is preceded by whitespace, the marker is removed.
- If it is preceded by non-whitespace, the marker is replaced by a single colon.

That way, the second sentence in the above example's first paragraph would be rendered as "The next paragraph is a code sample:".

Tables

For *grid tables* (ref), you have to "paint" the cell grid yourself. They look like this:

```
+------------------------+------------+----------+----------+
| Header row, column 1   | Header 2   | Header 3 | Header 4 |
| (header rows optional) |            |          |          |
+========================+============+==========+==========+
| body row 1, column 1   | column 2   | column 3 | column 4 |
+------------------------+------------+----------+----------+
| body row 2             | ...        | ...      |          |
+------------------------+------------+----------+----------+
```

Simple tables (ref) are easier to write, but limited: they must contain more than one row, and the first column cells cannot contain multiple lines. They look like this:

```
=====  =====  =======
A      B      A and B
=====  =====  =======
False  False  False
True   False  False
False  True   False
True   True   True
=====  =====  =======
```

Two more syntaxes are supported: *CSV tables* and *List tables*. They use an *explicit markup block*, see Directives section.

Hyperlinks

External links

Use `Link text <http://example.com/>`_ for inline web links. If the link text should be the web address, you don't need special markup at all, the parser finds links and mail addresses in ordinary text.

Important

There must be a space between the link text and the opening < for the URL.

You can also separate the link and the target definition (ref), like this:

```
This is a paragraph that contains `a link`_.

.. _a link: http://example.com/
```

Internal links

Internal linking is done via a special reST role provided by Sphinx, see the section on specific markup, Cross-referencing arbitrary locations.

Sections

Section headers (ref) are created by underlining (and optionally overlining) the section title with a punctuation character, at least as long as the text:

```
=================
This is a heading
=================
```

Normally, there are no heading levels assigned to certain characters as the structure is determined from the succession of headings. However, this convention is used in Python's Style Guide for documenting which you may follow:

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

Of course, you are free to use your own marker characters (see the reST documentation), and use a deeper nesting level, but keep in mind that most target formats (HTML, LaTeX) have a limited supported nesting depth.

Explicit Markup

"Explicit markup" (ref) is used in reST for most constructs that need special handling, such as footnotes, specially-highlighted paragraphs, comments, and generic directives.

An explicit markup block begins with a line starting with `..` followed by whitespace and is terminated by the next paragraph at the same level of indentation. (There needs to be a blank line between explicit markup and normal paragraphs. This may all sound a bit complicated, but it is intuitive enough when you write it.)

Directives

A directive (ref) is a generic block of explicit markup. Besides roles, it is one of the extension mechanisms of reST, and Sphinx makes heavy use of it.

Docutils supports the following directives:

- Admonitions: attention, caution, danger, error, hint, important, note, tip, warning and the generic admonition. (Most themes style only "note" and "warning" specially.)
- Images:
  - image (see also Images below)
  - figure (an image with caption and optional legend)
- Additional body elements:
  - contents (a local, i.e. for the current file only, table of contents)
  - container (a container with a custom class, useful to generate an outer `<div>` in HTML)
  - rubric (a heading without relation to the document sectioning)
  - topic, sidebar (special highlighted body elements)
  - parsed-literal (literal block that supports inline markup)
  - epigraph (a block quote with optional attribution line)
  - highlights, pull-quote (block quotes with their own class attribute)
  - compound (a compound paragraph)
- Special tables:
  - table (a table with title)
  - csv-table (a table generated from comma-separated values)
  - list-table (a table generated from a list of lists)
- Special directives:
  - raw (include raw target-format markup)
  - include (include reStructuredText from another file) – in Sphinx, when given an absolute include file path, this directive takes it as relative to the source directory
  - class (assign a class attribute to the next element) [1]
- HTML specifics:
  - meta (generation of HTML `<meta>` tags)
  - title (override document title)
- Influencing markup:
  - default-role (set a new default role)

- role (create a new role)

Since these are only per-file, better use Sphinx's facilities for setting the **default_role**.

Do *not* use the directives sectnum, header and footer.

Directives added by Sphinx are described in Sphinx Markup Constructs.

Basically, a directive consists of a name, arguments, options and content. (Keep this terminology in mind, it is used in the next chapter describing custom directives.) Looking at this example,

```
.. function:: foo(x)
              foo(y, z)
   :module: some.module.name

   Return a line of text input from the user.
```

`function` is the directive name. It is given two arguments here, the remainder of the first line and the second line, as well as one option `module` (as you can see, options are given in the lines immediately following the arguments and indicated by the colons). Options must be indented to the same level as the directive content.

The directive content follows after a blank line and is indented relative to the directive start.

Images

reST supports an image directive (ref), used like so:

```
.. image:: gnu.png
   (options)
```

When used within Sphinx, the file name given (here `gnu.png`) must either be relative to the source file, or absolute which means that they are relative to the top source directory. For example, the file `sketch/spam.rst` could refer to the image `images/spam.png` as `../images/spam.png` or `/images/spam.png`.

Sphinx will automatically copy image files over to a subdirectory of the output directory on building (e.g. the `_static` directory for HTML output.)

Interpretation of image size options (`width` and `height`) is as follows: if the size has no unit or the unit is pixels, the given size will only be respected for output channels that support pixels. Other units (like `pt` for points) will be used for HTML and LaTeX output (the latter replaces `pt` by `bp` as this is the TeX unit such that `72bp=1in`).

Sphinx extends the standard docutils behavior by allowing an asterisk for the extension:

```
.. image:: gnu.*
```

Sphinx then searches for all images matching the provided pattern and determines their type. Each builder then chooses the best image out of these candidates. For instance, if the file name `gnu.*` was given and two files `gnu.pdf` and `gnu.png` existed in the source tree, the LaTeX builder would choose the former, while the HTML builder would prefer the latter. Supported image types and choosing priority are defined at Available builders.

Note that image file names should not contain spaces.

Changed in version 0.4: Added the support for file names ending in an asterisk.

Changed in version 0.6: Image paths can now be absolute.

Changed in version 1.5: latex target supports pixels (default is `96px=1in`).

Footnotes

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

```
Lorem ipsum [#f1]_ dolor sit amet ... [#f2]_
```

```
.. rubric:: Footnotes

.. [#f1] Text of the first footnote.
.. [#f2] Text of the second footnote.
```

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

Citations

Standard reST citations (ref) are supported, with the additional feature that they are “global”, i.e. all citations can be referenced from all files. Use them like so:

```
Lorem ipsum [Ref]_ dolor sit amet.

.. [Ref] Book or article reference, URL or whatever.
```

Citation usage is similar to footnote usage, but with a label that is not numeric or begins with `#`.

Substitutions

reST supports “substitutions” (ref), which are pieces of text and/or markup referred to in the text by `|name|`. They are defined like footnotes with explicit markup blocks, like this:

```
.. |name| replace:: replacement *text*
```

or this:

```
.. |caution| image:: warning.png
              :alt: Warning!
```

See the reST reference for substitutions for details.

If you want to use some substitutions for all documents, put them into **rst_prolog** or put them into a separate file and include it into all documents you want to use them in, using the **include** directive. (Be sure to give the include file a file name extension differing from that of other source files, to avoid Sphinx finding it as a standalone document.)

Sphinx defines some default substitutions, see Substitutions.

Comments

Every explicit markup block which isn’t a valid markup construct (like the footnotes above) is regarded as a comment (ref). For example:

```
.. This is a comment.
```

You can indent text after a comment start to form multiline comments:

```
..
   This whole indented block
   is a comment.

   Still in the comment.
```

Source encoding

Since the easiest way to include special characters like em dashes or copyright signs in reST is to directly write them as Unicode characters, one has to specify an encoding. Sphinx assumes source files to be encoded in UTF-8 by default; you can change this with the **source_encoding** config value.

Gotchas

There are some problems one commonly runs into while authoring reST documents:

- **Separation of inline markup:** As said above, inline markup spans must be separated from the surrounding text by non-word characters, you have to use a backslash-escaped space to get around that. See the reference for the details.
- **No nested inline markup:** Something like `*see :func:`foo`*` is not possible.

Footnotes

[1] When the default domain contains a **`class`** directive, this directive will be shadowed. Therefore, Sphinx re-exports it as **`rst-class`**.