rinohtype

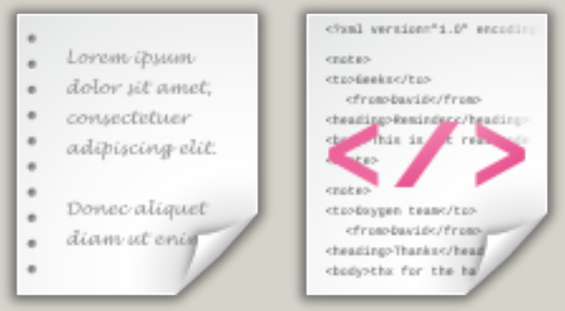
the Python document processor

Brecht Machiels

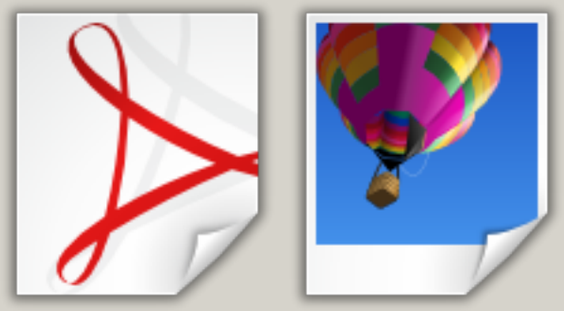
BATCH-MODE DOCUMENT PROCESSOR

Structured Document

reStructuredText,
Markdown,
DocBook, DITA

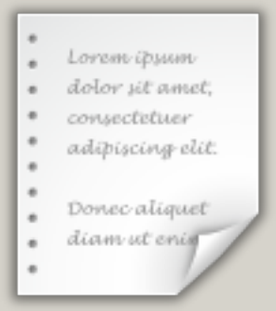


Images
PDF, PNG, JPEG



Style Definition

Style Sheet and
Document Template
Configuration (INI)



Fonts
OpenType,
TrueType, Type1

Output

professionally
typeset paged
document (PDF)

USAGE

- command line tool

```
$ rinoh README.rst
...
Writing output: README.pdf
```

```
$ rinoh --paper A5 my_document.md
...
42% [=============                    ] ETA 00:04 (00:03) page 6
```

- Sphinx builder: drop-in replacement for LaTeX builder

```
$ sphinx-build -b rinoh sourcedir outdir
...
Writing output: output/manual.pdf
```

INSTALLATION

- Python 3.3 and up

```
$ pip install rinohtype
...
Successfully installed commonmark-0.5.4 docutils-0.13.1
purepng-0.2.0 recommonmark-0.4.0 rinoh-typeface-
dejavuserif-0.1.1 rinoh-typeface-texgyrecursor-0.1.1 rinoh-
typeface-texgyreheros-0.1.1 rinoh-typeface-
texgyrepagella-0.1.1 rinohtype-0.3.1
```

- Stand-alone application (next release)
  - Windows installer
  - macOS app bundle

DEMO (1)

- render a reStructuredText document

```
$ rinoh example.rst
```

  - produces `example.pdf`
  - options: see `rinoh --help`

- build a Sphinx project
  - add rinoh to `extensions`
  - use `latex_documents`

DOCUMENT TREE

1 Section 1

This is a paragraph. This sentence contains a link to the Python homepage.

1.1 Subsection

Column 1		Column 2
Cell 1	Cell 2	Cell 3

2 Section 2

Next up, an enumerated list!

1. a list item
2. this list item contains a bulleted list
   - first bullet item
   - second bullet item

```
Section(id='section-1')
  Heading("1 Section 1")
  Paragraph("This is a paragraph…")
  Section(id='subsection')
    Heading("1.1 Subsection")
    Table()
      TableHead()
        TableRow()
          TableCell()
            Paragraph("Column 1")
          TableCell()
      TableBody()
Section(id='section-2')
  Heading("2 Section 2")
  Paragraph("Next up, an enumerated…")
  List(style='enumerated')
    ListItem()
    ListItem()
      ListItemLabel("2.")
      StaticGroupedFlowables()
        Paragraph("this list item…")
        List(style='bulleted')
          ListItem()
            ListItemLabel("•")
            StaticGroupedFlowables()
              Paragraph("first…")
          ListItem()
```

INLINE ELEMENTS

Text with *multiple* ***nested*** ***STYLES***.

```
Paragraph
├─ SingleStyledText("Text with ")
├─ MixedStyledText(style='emphasis')
│  ├─ SingleStyledText("multiple ")
│  └─ MixedStyledText(style='strong')
│     ├─ SingleStyledText("nested ")
│     └─ SingleStyledText("styles", style='small caps')
└─ SingleStyledText(".")
```

SELECTORS

```
# match based on context

TableCell / Paragraph     ①

List / ... / Paragraph    ②


# match based on style

List.like('enumerated')   ③


# match arbitrary attributes

Section.like(level=2) / Heading   ④

TableCell.like(row_index=slice(0, None, 2),
               rowspan=1)
```

```
Section(id='section-1')
  Heading("1 Section 1")
  Paragraph("This is a paragraph…")
  Section(id='subsection')
    Heading("1.1 Subsection")   ④
    Table()
      TableHead()
        TableRow()
          TableCell()
            Paragraph("Column 1")   ①
          TableCell()
      TableBody()
Section(id='section-2')
  Heading("2 Section 2")
  Paragraph("Next up, an enumerated…")
  List(style='enumerated')   ③
    ListItem()
    ListItem()
      ListItemLabel("2.")
      StaticGroupedFlowables()
        Paragraph("this list item…") ②
        List(style='bulleted')
          ListItem()
            ListItemLabel("•")
            StaticGroupedFlowables()
              Paragraph("first…")   ②
          ListItem()
```

MATCHER

- maps style names to selectors

style name	selector
emphasized text	`StyledText.like('emphasis')`
line block	`GroupedFlowables.like('line block')`
nested line block	`'line block' / 'line block'`
enumerated list	`List.like('enumerated')`
enumerated list item	`'enumerated list' / ListItem`
figure legend	`Figure / GroupedFlowables.like('legend')`

STYLE SHEET

```
[STYLESHEET]
name = My style

[VARIABLES]
sans_typeface = Helvetica
blue = #20435c

[default:Paragraph]
typeface = $(sans_typeface)
font_weight = regular
font_size = 10pt
line_spacing = fixed(12pt, leading(0))
text_align = justify
kerning = true
ligatures = true

[body]
base = default
space_above = 5pt

[emphasis]
font_slant = italic
```

STYLE SHEET – INHERITANCE

```
[STYLESHEET]
name = My Sphinx style
description = Sphinx style sheet tweaks
base = sphinx

[VARIABLES]
sans_typeface = Droid Sans

[emphasis]
font_color = #00a

[strong]
base = DEFAULT_STYLE
font_color = #a00
```

DEMO (2)

- tweak the document style
  - style log: which style definition?
  - create a custom style sheet
    - extending the default
  - override
    - the serif typeface
    - the color of section titles

```
$ rinoh --stylesheet my_style.rst example.rst
```

DEMO (2)

1 Section 1

This is a paragraph. This sentence contains a link to the Python homepage.

1.1 Subsection

Column 1		Column 2
Cell 1	Cell 2	Cell 3

2 Section 2

Next up, an enumerated list!

1. a list item
2. this list item contains a bulleted list
    - first bullet item
    - second bullet item

I. Section 1

This is a paragraph. This sentence contains a link to the Python homepage.

I.A Subsection

Column 1		Column 2
Cell 1	Cell 2	Cell 3

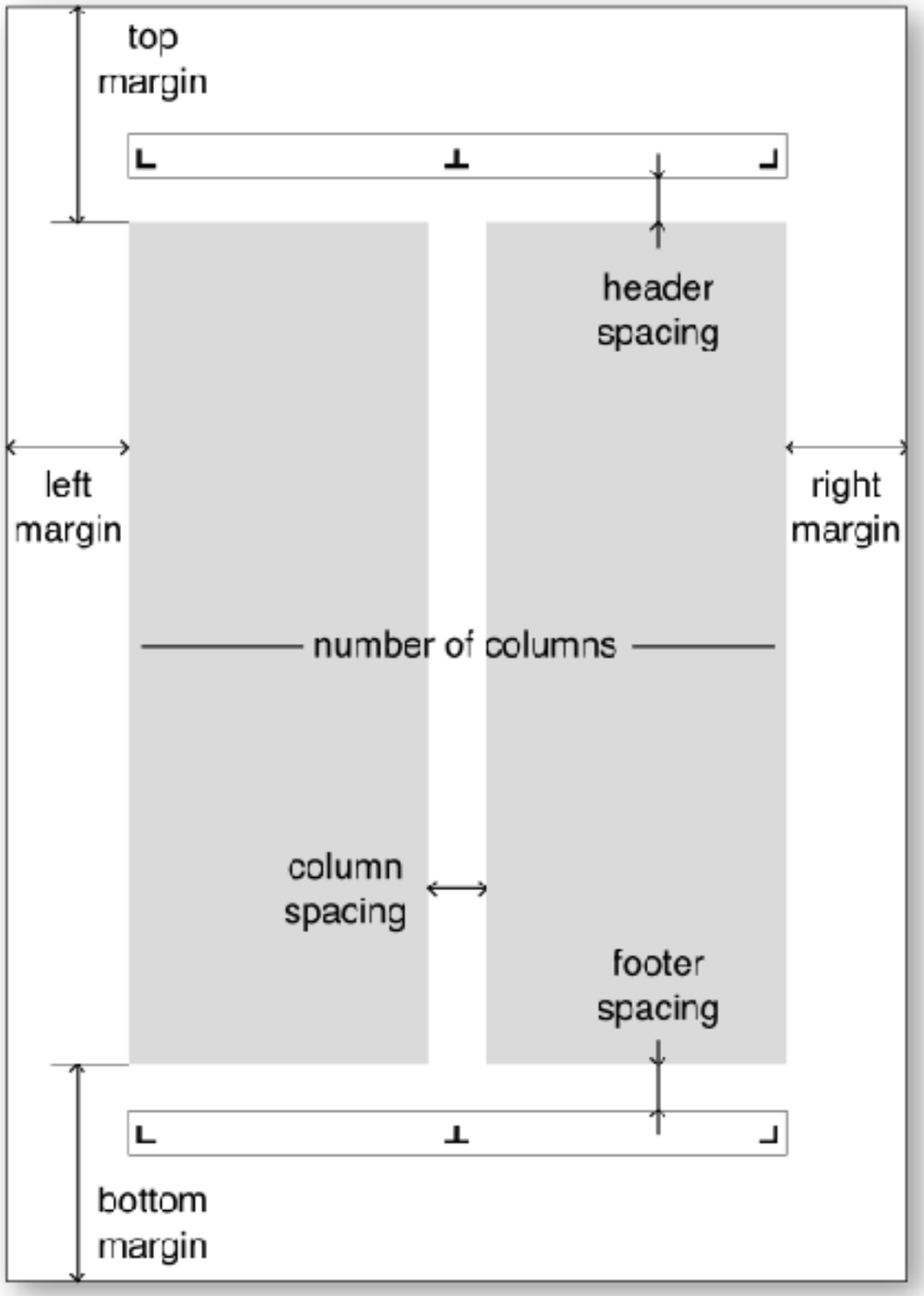
II. Section 2

Next up, an enumerated list!

1. a list item
2. this list item contains a bulleted list
    - first bullet item
    - second bullet item

DOCUMENT TEMPLATES

- document parts
  - title page
  - front matter
    - table of contents
  - contents
  - back matter
    - index
- page templates



TEMPLATE CONFIGURATION

```
[TEMPLATE_CONFIGURATION]
name = my article configuration
template = article

parts = title contents
stylesheet = my_style.rts
language = fr
abstract_location = title

[SectionTitles]
contents = 'Contents'

[VARIABLES]
paper_size = A5

[title]
page_number_format = lowercase roman
end_at_page = left

[title_page]
background = 'title.pdf' scale=fill
top_margin = 2cm
```

Quickstart 3

This section gets you started quickly, discussing each of the three modes of operation introduced in Introduction. If you want to customize the style of the PDF document, please refer to Basic Document Styling which introduces style sheets and document templates.

3.1 Command-Line Renderer

Installing rinohtype places the rinoh script in the PATH. This can be used to render reStructuredText documents such as demo.txt:

```
rinoh --format reStructuredText demo.txt
```

After rendering finishes, you will find demo.pdf alongside the input file.

rinoh allows specifying the document template and style sheet to use when rendering the reStructuredText document. See its command-line options for details.

Two rendering passes are required to make sure that cross-references to page numbers are correct. After a document has been rendered, rinohtype will save the page reference data to a .rtc file. Provided the document (or the template or style sheet) doesn't change a lot, this can prevent the need to perform a second rendering pass.

3.2 Sphinx Builder

To use rinohtype to render Sphinx documents, at a minimum you need to add 'rinoh.frontend.sphinx' to the extensions list in the Sphinx project's conf.py.

If your Sphinx project is already configured for rendering with LaTeX, rinohtype will happily interpret latex_documents and other options for the LaTeX builder. Otherwise, you need to set the rinoh_documents configuration option:

```
rinoh_documents = [('index',            # top-level file (index.rst)
                    'target',           # output (target.pdf)
                    'Document Title',   # document title
                    'John A. Uthor')]   # document author
```

Other configuration variables are optional and allow configuring the style of the generated PDF document. See Sphinx Builder for details.

When building the documentation, select the rinoh builder by passing it to sphinx-build -b option:

```
sphinx-build -b rinoh . _build/rinoh
```

5.3 Matchers

At the most basic level, a StyledMatcher is a dictionary that maps labels to selectors:

```
matcher = StyledMatcher()
...
matcher['emphasis'] = StyledText.like('emphasis')
matcher['chapter'] = Section.like(level=1)
matcher['list item number'] = ListItem / Paragraph
matcher['nested line block'] = (GroupedFlowables.like('line block')
                                / GroupedFlowables.like('line block'))
...
```

Rinohtype currently includes one matcher which defines labels for all common elements in documents:

```
from rinoh.stylesheets import matcher
```

5.4 Style Sheets

A StyleSheet takes a StyledMatcher to provide element labels to assign style properties to:

```
styles = StyleSheet('IEEE', matcher=matcher)
...
styles['strong'] = TextStyle(font_weight=BOLD)
styles('emphasis', font_slant=ITALIC)
styles('nested line block', margin_left=0.5*CM)
...
```

Each Styled has a Style class associated with it. For Paragraph, this is ParagraphStyle. These style classes determine which style attributes are accepted for the styled element. Because the style class can automatically be determined from the selector, it is possible to simply pass the style properties to the style sheet by calling the StyleSheet instance as shown above.

Style sheets are usually loaded from a .rts file using StyleSheetFile. An example style sheet file is shown in Style Sheets.

A style sheet file contains a number of sections, denoted by a section title enclosed in square brackets. There are two special sections:

- [STYLESHEET] describes global style sheet information (see StyleSheetFile for details)
- [VARIABLES] collects variables that can be referenced elsewhere in the style sheet

Other sections define the style for a document elements. The section titles correspond to the labels associated with selectors in the StyledMatcher. Each entry in a section sets a value for a style attribute. The style for enumerated lists is defined like this, for example:

```
[enumerated list]
margin_left=8pt
space_above=5pt
space_below=5pt
ordered=true
flowable_spacing=5pt
number_format=NUMBER
```



```
label_suffix=')'
```

Since this is an enumerated list, ordered is set to true. number_format and label_suffix are set to produce list items labels of the style 1), 2), ... Other entries control margins and spacing. See ListStyle for the full list of accepted style attributes.

5.4.1 Base Styles

It is possible to define styles which are not linked to a selector. These can be useful to collect common attributes in a base style for a set of style definitions. For example, the Sphinx style sheet defines the header_footer style to serve as a base for the header and footer styles:

```
[header_footer : Paragraph]
base=default
typeface=$(sans_typeface)
font_size=10pt
font_weight=BOLD
indent_first=0pt
tab_stops=50% CENTER, 100% RIGHT

[header]
base=header_footer
padding_bottom=2pt
border_bottom=$(thin_black_stroke)
space_below=24pt

[footer]
base=header_footer
padding_top=4pt
border_top=$(thin_black_stroke)
space_above=18pt
```

Because there is no selector associated with header_footer, the element type needs to be specified manually. This is done by adding the name of the relevant Styled subclass to the section name, using a colon (:) to separate it from the style name, optionally surrounded by spaces.

5.4.2 Custom Selectors

It is also possible to define new selectors directly in a style sheet file. This allows making tweaks to an existing style sheet without having to create a new StyledMatcher. However, this should be used sparingly. If a great number of custom selectors are required, it is better to create a new StyledMatcher

The syntax for specifying a selector for a style is similar to that when constructing selectors in a Python source code (see Matchers), but with a number of important differences. A Styled subclass name followed by parentheses represents a simple class selector (without context). Arguments to be passed to Styled.like() can be included within the parentheses.

```
[special text : StyledText('special')]
font_color=#FF00FF

[accept button : InlineImage(filename='images/ok_button.png')]
baseline=20%
```

Even if no arguments are passed to the class selector, it is important that the class name is followed by parentheses. If the parentheses are omitted, the selector is not registered with the matcher and the style can only be used as a base style for other style definitions (see Base Styles).

RESOURCES

- find by name (entry point)
  - style sheets
  - document templates
  - typefaces (fonts)
  - frontends
    - DITA (commercial)

```
rinoh --list-stylesheets

rinoh --list-templates

rinoh --list-fonts

rinoh --list-formats
```

- future: auto-install from PyPI

THANK YOU

- http://www.mos6581.org/rinohtype/
- https://github.com/brechtm/rinohtype

- Brecht Machiels
  - email: brecht@mos6581.org
  - blog: http://www.mos6581.org
  - available for consultancy projects!