rinohtype

Release 0.2.1

Brecht Machiels

August 18, 2016

Table of Contents

Release v0.2.1. (*Release History*)

Rinohtype is a high-level PDF library for Python. It helps automating the creation of any type of document, ranging from invoices to long, complex technical documents.

Paired with Sphinx, rinohtype offers a modern alternative to LaTeX . Sphinx helps writing large structured documents and supports a multitude of different output formats including searchable HTML. Rinohtype provides a Sphinx backend that allows generating beautiful PDF documents.

Here is a list of rinohtype's main features:

- a powerful page layout system supporting columns, running headers/footers, floatable elements and footnotes
- figures, large tables and automatically generated table of contents
- automatic numbering and cross-referencing of headings, figures and tables
- use one of the included document templates or create your own
- an intuitive style sheet system inspired by CSS
- modular design allowing for multiple frontends (such as reStructuredText, Markdown, DocBook, ...) and backends (PDF, SVG, bitmap, ...)
- handles OpenType, TrueType and Type1 fonts with support for advanced typographic features such as kerning, ligatures and small capitals
- embeds PDF, PNG and JPEG images, preserving transparency and color profiles
- easy to deploy; pure-Python with few dependencies
- built on Unicode; ready for non-latin languages

Rinohtype is currently in a beta phase. We are working toward a first stable release.

Rinohtype is open source software licensed under the GNU AGPL 3.0. Practically, this means you are free to use it in open-source software, but not in (commercial) closed-source software. For this purpose, you need a commercial license (see http://www.opqode.com/). We are also available for consultancy projects involving rinohtype, so please don't hesitate to contact us.

CHAPTER 1

Introduction

Rinohtype was initially conceived as a modern replacement for LaTeX. An important goal in the design of rinohtype is for documents to be much easier to customize than in LaTeX. By today's standards, the arcane TeX macro language upon which LaTeX is built makes customization unnecessarily difficult for one. Simply being built with Python makes rinohtype already much easier to approach than TeX. Additionally, rinohtype is built around the following core concepts to ensure customizability:

Document templates
: These determine the page layout and (for longer documents) the different parts of your document.

Style sheets
: The CSS-inspired style sheets determine the look of individual document elements. A style sheet assigns style attributes to each type of document element. For example, a paragraph's style is determined by the typeface, font weight, size and color, horizontal alignment of text etc.

Structured Input
: Rinohtype renders a document from a document tree that does not describe any style aspects but only semantics. The style sheet maps specific style properties to the elements in this document tree. The document tree can be generated from a structured document format such as reStructuredText and DocBook using one of the included frontends, or it can be built manually.

Rinohtype is implemented as a Python package and doubles as a high-level PDF library. Its modular design makes it easy to to customize and extend for specific applications. Because rinotype's source code is open, all of its internals can be inspected and even modified, making it extremely customizable.

1.1 Usage Examples

Rinohtype supports three modes of operation. These are discussed in more detail in the *Quickstart* guide.

For each of these modes, you can choose to use one of the document templates included with rinohtype or a third-party template available from PyPI and optionally customize it to your needs. Or you can create a custom template from scratch. The same goes for the style sheet used to style the document elements.

1.1.1 reStructuredText Renderer

Rinohtype includes the **`rinoh`** command-line tool that can render reStructuredText documents. Rendering the reStructuredText demonstration article demo.txt (using the standard article template and style sheet) generates `demo.pdf`.

1.1.2 Sphinx Builder

Configuring rinohtype as a builder for Sphinx allows rendering a Sphinx project to PDF without the need

for a LaTeX installation. This very document you are reading was rendered using rinohtype's Sphinx builder.

1.1.3 High-level PDF library

Rinohtype can be used as a Python library to generate PDF documents. Just like with **`rinoh`** and the Sphinx builder, you can select which document template and style sheet to use.

Additionally, you need to supply a document tree. This document tree can be parsed from a structured document format such as reStructuredText by using one of the provided frontends or built manually using building blocks provided by rinohtype. You can also write a frontend for a custom format such as an XML dialect.

All of these approaches allow for parts of the content to be fetched from a database or other data sources. When parsing the document tree from a structured document format, a templating engine like Jinja2 can be used.

CHAPTER 2

Installation

Rinohtype supports Python 3.3 and up. Depending on demand, it might be back-ported to Python 2.7, however[1].

Use pip to install the latest version of rinohtype and its dependencies:

```
pip install rinohtype
```

If you plan on using rinohtype as an alternative to LaTeX, you will want to install Sphinx as well:

```
pip install Sphinx
```

See *Sphinx Builder* in the *Quickstart* guide on how to render Sphinx documents with rinohtype.

2.1 Dependencies

For parsing reStructuredText documents, rinohtype depends on docutils. For parsing PNG images the pure-Python PurePNG package is required. pip takes care of these requirements automatically when you install rinohtype.

If you want to include images other than PDF, PNG or JPEG, you will need to install Pillow additionally.

1 Be sure to contact us if you are interested in running rinohtype on Python 2.7.

CHAPTER 3

Quickstart

This section gets you started quickly, discussing each of the three modes of operation introduced in *Introduction*. Additionally, the basics of style sheets and document templates are explained.

3.1 reStructuredText Renderer

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

```
rinoh 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**'s `-b` option:

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

Just like the **rinoh** command line tool, the Sphinx builder requires two *rendering passes*.

3.3 High-level PDF Library

Note:

The focus of rinohtype development is currently on the **rinoh** tool and Sphinx builder at this moment. Use as a Python library is possible, but documentation may be lacking. Please be patient.

The most basic way to use rinohtype in an application is to hook up an included frontend, a document template and a style sheet:

```
from rinoh.backend import pdf
from rinoh.frontend.rst import ReStructuredTextReader
from rinoh.templates import Article

# the parser builds a rinohtype document tree
parser = ReStructuredTextReader()
with open('my_document.rst') as file:
    document_tree = parser.parse(file)

# render the document to 'my_document.pdf'
document = Article(document_tree, backend=pdf)
document.render('my_document')
```

This basic application can be customized to your specific requirements by customizing the document template, the style sheet and the way the document's content tree is built. The basics of document templates and style sheets are covered the the sections below.

The document tree returned by the `ReStructuredTextReader` in the example above can also be built manually. A `DocumentTree` is simply a list of `Flowable`s, which can have child elements. These children in turn can also have children, and so on; together they form a tree.

Here is an example document tree of a short article:

```
from rinoh.document import DocumentTree
from rinoh.styleds import *

document_tree = DocumentTree('/path/to/source_file.ext',
                [Paragraph('My Document', style='title'), # metadata!
                 Section([Heading('First Section'),
                          Paragraph('This is a paragraph with some'
                                    + Emphasized('emphasized text')
                                    + 'and an'
                                    + InlineImage('image.pdf')),
                          Section([Heading('A subsection'),
                                   Paragraph('Another paragraph')
                                  ])
                         ]),
                 Section([Heading('Second Section'),
                          List([Paragraph('a list item'),
                                Paragraph('another list item')
                               ])
                         ])
```

```
                ])
```

It is clear that this type of content is best parsed from a structured document format such as reStructuredText or XML. Manually building a document tree is well suited for short, custom documents however.

3.4 Style Sheets

A style sheet determines the look of each of the elements in a document. For each type of document element, the style sheet object registers a list of style properties. Style sheets are stored in plain text files using the INI format with the `.rts` extension. Below is an excerpt from the *Sphinx* style sheet included with rinohtype.

```
[STYLESHEET]
name=Sphinx
description=Mostly a copy of the LaTeX style included with Sphinx
pygments_style=friendly

[VARIABLES]
mono_typeface=TeX Gyre Cursor
serif_typeface=TeX Gyre Pagella
sans_typeface=Tex Gyre Heros
thin_black_stroke=0.5pt,#000
blue=#20435c

[default:Paragraph]
typeface=$(serif_typeface)
font_weight=REGULAR
font_size=10pt
line_spacing=fixed(12pt, leading(0))
indent_first=0
space_above=0
space_below=0
text_align=JUSTIFY
kerning=True
ligatures=True
hyphen_lang=en_US
hyphen_chars=4

[body]
base=default
space_above=5pt
space_below=0
text_align=justify

[emphasis]
font_slant=italic

[strong]
font_weight=BOLD

[literal strong]
base=strong
typeface=$(mono_typeface)

[quote]
font_slant=italic
```

Except for `[STYLESHEET]` and `[VARIABLES]`, each configuration section in a style sheet determines the style of a particular type of document element. The `emphasis` style, for example, determines the look of emphasized text, which is displayed in an italic font. For more on style sheets, please refer to the *Styling Document Elements* section in *Advanced Topics*.

This is similar to how HTML's cascading style sheets work. In rinohtype however, document elements are identified by means of a descriptive label (such as *emphasis*) instead of a cryptic selector. Rinohtype also makes use of selectors, but these are collected in a `StyledMatcher` which maps them to descriptive names to be used by many style sheets. Unless you are using rinohtype as a PDF library to create custom documents, the default matcher should cover your needs.

3.4.1 Extending an Existing Style Sheet

Starting from an existing style sheet, it is easy to make small changes to the style of individual document elements. The following example creates a new style sheet based on the Sphinx stylesheet included with rinohtype. The style sheet redefines the style for emphasized text, displaying it in a bold instead of italic font.

```
[STYLESHEET]
name=My Style Sheet
description=Small tweaks made to the Sphinx style sheet
base=sphinx

[VARIABLES]
mono_typeface=Courier

[emphasis]
font_slant=bold
```

This style sheet also redefines the `mono_typeface` variable. This variable is used in the Sphinx style sheet in all style definitions where a monospaced font is desired. Redefining the variable affects all of these style definitions.

To use this style sheet, load it using `StyleSheetFile`:

```
from rinoh.style import StyleSheetFile

my_stylesheet = StyleSheetFile('m_stylesheet.rts')
```

3.4.2 Starting from Scratch

If you don't specify a base style sheet in the `[STYLESHEET]` section, you create an independent style sheet. You should do this if you want to create a document style that is radically different from what is provided by existing style sheets.

If the style definition for a particular document element is missing, the default values for its style properties are used.

To use this style sheet, you need to specify the `StyledMatcher` to use, since there is not base providing one. The following example uses the standard matcher:

```
from rinoh.style import StyleSheetFile
from rinoh.stylesheets import matcher

my_stylesheet = StyleSheetFile('m_stylesheet.rts', matcher=matcher)
```

Note:

In the future, rinohtype will allow generating an INI style sheet, listing all matched elements and their style attributes together with the default values.

3.5 Document Templates

As with style sheets, you can choose to make use of the templates provided by rinohtype and optionally customize it or you can create a custom template from scratch.

3.5.1 Using an Existing Template

Rinohtype provides a number of *Standard Document Templates*. These can be customized by passing an instance of the associated `rinoh.template.TemplateConfiguration` as *configuration* on template instantiation.

The example from *High-level PDF Library* above can be customized by setting template options.

```
from rinoh.backend import pdf
from rinoh.dimension import CM
from rinoh.frontend.rst import ReStructuredTextReader
from rinoh.paper import A5
from rinoh.stylesheets import sphinx_base14
from rinoh.templates import Article

# the parser builds a rinohtype document tree
parser = ReStructuredTextReader()
with open('my_document.rst') as file:
    document_tree = parser.parse(file)

# customize the article template
configuration = Article.Configuration(paper_size=A5,
                                      stylesheet=sphinx_base14,
                                      abstract_location='title',
                                      table_of_contents=False)
configuration('title_page', top_margin=2*CM)

# render the document to 'my_document.pdf'
document = Article(document_tree, configuration=configuration, backend=pdf)
document.render('my_document')
```

Here, a number of global template settings are specified to override the default values. For example, the paper size is changed to A5 from the default A4. See below for a list of the settings that can be changed and their description.

The top margin of the title page is also changed by setting the corresponding option for the `rinoh.templates.article.ArticleConfiguration.title_page` page template.

3.5.2 Creating a Custom Template

A custom template can be created by inheriting from `DocumentTemplate`. The `parts` attribute determines the global structure of the document. It is a list of `DocumentPartTemplate`s, each referencing a page template.

The `rinoh.templates.article.Article` template, for example, consists of a title page, a front matter part (a custom `rinoh.template.DocumentPartTemplate` subclass) and the article contents:

```
class Article(DocumentTemplate):
```

```
    Configuration = ArticleConfiguration
    parts = [TitlePartTemplate('title', Configuration.title_page),
             ArticleFrontMatter('front matter', Configuration.page),
             ContentsPartTemplate('contents', Configuration.page)]
```

The `TemplateConfiguration` subclass associated with `Article`

- overrides the default style sheet,
- introduces configuration attributes to control the table of contents placement and the location of the abstract, and
- defines the page templates referenced in `Article.parts`

```
class ArticleConfiguration(TemplateConfiguration):
    stylesheet = OverrideDefault(sphinx_article)
    table_of_contents = Attribute(Bool, True,
                                  'Show or hide the table of contents')
    abstract_location = Attribute(AbstractLocation, FRONT_MATTER,
                                  'Where to place the abstract')

    title_page = TitlePageTemplate(page_size=Var('paper_size'),
                                   top_margin=8*CM)
    page = PageTemplate(page_size=Var('paper_size'),
                        chapter_title_flowables=None)
```

The configuration attributes are used in the custom front matter document part template:

```
class ArticleFrontMatter(DocumentPartTemplate):
    def flowables(self, document):
        meta = document.metadata
        abstract_loc = document.configuration.get_option('abstract_location')
        if 'abstract' in meta and abstract_loc == FRONT_MATTER:
            yield meta['abstract']
        if document.configuration.get_option('table_of_contents'):
            yield TableOfContentsSection()
```

CHAPTER 4

Command-line Renderer

4.1 rinoh

Render a reStructuredText document to PDF.

usage: rinoh [-h] [-t [TEMPLATE]] [-s [NAME OR FILENAME]] [-p [PAPER]] [--list-templates] [--list-stylesheets] [--version] [input]

input
the reStructuredText document to render

-h, **--help**
show this help message and exit

-t <template>, **--template** <template>
the document template to use (default: article)

-s <name or filename>, **--stylesheet** <name or filename>
the style sheet used to style the document elements (default: the template's default)

-p <paper>, **--paper** <paper>
the paper size to render to (default: A4)

--list-templates
list the installed document templates and exit

--list-stylesheets
list the installed style sheets and exit

--version
show program's version number and exit

CHAPTER 5

Sphinx Builder

The `rinoh.frontent.sphinx` is a Sphinx extension module. It provides a Sphinx builder with the name *rinoh*.

The *rinoh* builder recognizes the following `conf.py` options. Of thse, only *`rinoh_documents`* (or `latex_documents`) needs to be specified:

`rinoh_documents`
: Determines how to group the document tree into PDF output files. Its format is identical to that of `latex_documents`, with the exception that *targetname* should specify the name of the PDF file without the extension. If it is not specified, the value of `latex_documents` is used instead (with the `.tex` extension stripped from the *targetname*).

`rinoh_document_template`
: The document template to use for rendering the Sphinx documentation. It can be a `DocumentTemplate` subclass or a string identifying an installed template. For the latter, run *`rinoh --list-templates`* to list the available templates. Default: `'book'`.

`rinoh_template_configuration`
: This variable allows configuring the document template specified in *`rinoh_document_template`*. Its value needs to be an instance of the `TemplateConfiguration` subclass associated with the specified document template class. Default: `None`.

`rinoh_paper_size`
: If no paper size is configured in *`rinoh_template_configuration`*, this determines the paper size used. This should be a `Paper` instance. A set of predefined paper sizes can be found in the `rinoh.paper` module. If not specified, the value of the `'papersize'` entry in `latex_elements` is converted to the equivalent `Paper`. If this is not specified, the value specified for `latex_paper_size` is used.

`rinoh_stylesheet`
: If *`rinoh_template_configuration`* does not specify a style sheet, this variable specifies the style sheet used to style the document elements. It can be a `StyleSheet` instance or a string identifying an installed style sheet. Default: the default style sheet for the chosen document template.

  If `pygments_style` is specified, it overrides the code highlighting style for the specified or default style sheet.

Note:

Since the interactions between *`rinoh_template_configuration`*, *`rinoh_paper_size`*,

rinoh_stylesheet and pygments_style are fairly complex, this behavior may be changed (simplified) in the future.

---

rinoh_logo
: Path (relative to the configuration directory) to an image file to use at the top of the title page. If not specified, the latex_logo value is used.

rinoh_domain_indices
: Controls the generation of domain-specific indices. Identical to latex_domain_indices, which is also used when *rinoh_domain_indices* is not specified.

CHAPTER 6

Standard Document Templates

Rinohtype includes a number of document templates. These are configurable and therefore should cater for most documents.

6.1 Article

The article template consists of a title page, an optional table of contents and the body text. By default, it uses a single numbering style for all pages and the same template for even and odd pages.

class rinoh.templates.article.**Article** (*document_tree*, *strings=None*, *configuration=None*, *options=None*, *backend=None*)

This template builds a document consisting of the following parts:

- **title** (*rinoh.template.TitlePartTemplate*)
  - *page_template*: *ArticleConfiguration.title_page*
  - *page_number_format*: NUMBER
- **front matter** (rinoh.templates.article.ArticleFrontMatter)
  - *page_template*: *ArticleConfiguration.page*
  - *page_number_format*: NUMBER
- **contents** (*rinoh.template.ContentsPartTemplate*)
  - *page_template*: *ArticleConfiguration.page*
  - *page_number_format*: NUMBER

Configuration

alias of *ArticleConfiguration*

class rinoh.templates.article.**ArticleConfiguration** (*base=None*, ***kwargs*)

Parameters

- **stylesheet** -- Overrides TemplateConfiguration default: StyleSheetFile(Sphinx (article))
- **table_of_contents** (*Bool*) -- Show or hide the table of contents. Default: True
- **abstract_location** (*AbstractLocation*) -- Where to place the abstract.

Default: front_matter

Inherited parameters

- *TemplateConfiguration*: **stylesheet**, **paper_size**

title_page

Defaults for *rinoh.template.TitlePageTemplate*:

- **top_margin** = `226.77pt`
- **page_size** = `Var('paper_size')`

page

Defaults for *rinoh.template.PageTemplate*:

- **page_size** = `Var('paper_size')`
- **chapter_title_flowables** = `None`

class `rinoh.templates.article.`**AbstractLocation**

Where to place the article's abstract
Accepts these options:

- 'title'
- 'front_matter'

6.2 Book

The book template consists of a title page, the table of contents, the body text and an index. The front matter pages are numbered using lowercase roman numerals. The template uses different templates for even and odd pages.

class `rinoh.templates.book.`**Book** (*document_tree*, *strings=None*, *configuration=None*, *options=None*, *backend=None*)

This template builds a document consisting of the following parts:

- **title** (*rinoh.template.TitlePartTemplate*)
  - *page_template*: *BookConfiguration.title_page*
  - *page_number_format*: `NUMBER`
- **front matter** (*rinoh.template.FixedDocumentPartTemplate*)
  - *flowables*: `[<rinoh.structure.TableOfContentsSection object at 0x11018b400>]`
  - *right_page_template*: *BookConfiguration.front_matter_right_page*
  - *left_page_template*: *BookConfiguration.front_matter_left_page*
  - *page_number_format*: `LOWERCASE ROMAN`
- **contents** (*rinoh.template.ContentsPartTemplate*)

- *right_page_template*: *BookConfiguration.content_right_page*
- *left_page_template*: *BookConfiguration.content_left_page*
- *page_number_format*: NUMBER

- **indices** (*rinoh.template.FixedDocumentPartTemplate*)
  - *flowables*: [<rinoh.index.IndexSection object at 0x111a126d8>]
  - *right_page_template*: *BookConfiguration.back_matter_right_page*
  - *left_page_template*: *BookConfiguration.back_matter_left_page*
  - *page_number_format*: NUMBER

Configuration
alias of *BookConfiguration*

class rinoh.templates.book.**BookConfiguration** (*base=None*, ***kwargs*)

Parameters **stylesheet** -- Overrides TemplateConfiguration default: StyleSheetFile(Sphinx)

Inherited parameters

- *TemplateConfiguration*: **stylesheet**, **paper_size**

page
Defaults for *rinoh.template.PageTemplate*:

- **top_margin** = 72.00pt
- **right_margin** = 72.00pt
- **bottom_margin** = 72.00pt
- **left_margin** = 72.00pt
- **page_size** = Var('paper_size')

title_page
Defaults for *rinoh.template.TitlePageTemplate*:

front_matter_right_page
Defaults for *rinoh.template.PageTemplate*:

- **chapter_header_text** = None
- **footer_text** = '\t\t$PAGE_NUMBER'
- **chapter_title_height** = 180.00pt
- **header_footer_distance** = 0
- **chapter_title_flowables** = <function front_matter_section_title_flowables at 0x111f58488>
- **chapter_footer_text** = '\t\t$PAGE_NUMBER'
- **header_text** = None

front_matter_left_page
Defaults for *rinoh.template.PageTemplate*:

- **header_footer_distance** = 0
- **footer_text** = '$PAGE_NUMBER'
- **header_text** = None

content_right_page
Defaults for *rinoh.template.PageTemplate*:

- **chapter_header_text** = None
- **footer_text** = '$SECTION_NUMBER(2). $SECTION_TITLE(2)\t\t$PAGE_NUMBER'
- **chapter_title_height** = 172.80pt
- **header_footer_distance** = 0
- **chapter_title_flowables** = <function body_matter_chapter_title_flowables at 0x111f580d0>
- **chapter_footer_text** = '\t\t$PAGE_NUMBER'
- **header_text** = '\t\t$DOCUMENT_TITLE, $DOCUMENT_SUBTITLE'

content_left_page
Defaults for *rinoh.template.PageTemplate*:

- **header_footer_distance** = 0
- **footer_text** = '$PAGE_NUMBER\t\tChapter $SECTION_NUMBER(1). $SECTION_TITLE(1)'
- **header_text** = '$DOCUMENT_TITLE, $DOCUMENT_SUBTITLE'

back_matter_right_page
Defaults for *rinoh.template.PageTemplate*:

- **chapter_header_text** = None
- **columns** = 2
- **footer_text** = '$SECTION_TITLE(1)\t\t$PAGE_NUMBER'
- **chapter_title_height** = 180.00pt
- **header_footer_distance** = 0
- **chapter_title_flowables** = <function front_matter_section_title_flowables at 0x111f58488>
- **chapter_footer_text** = '\t\t$PAGE_NUMBER'
- **header_text** = '\t\t$DOCUMENT_TITLE, $DOCUMENT_SUBTITLE'

back_matter_left_page
Defaults for *rinoh.template.PageTemplate*:

- **header_footer_distance** = 0
- **columns** = 2

- **footer_text** = `'$PAGE_NUMBER\t\t$SECTION_TITLE(1)'`
- **header_text** = `'$DOCUMENT_TITLE, $DOCUMENT_SUBTITLE'`

CHAPTER 7

Advanced Topics

This section serves as a guide to the internal workings of rinohtype, and more specifically on how element styling works. This will be helpful if you want to create custom style sheets or develop a new frontend.

7.1 Flowables and Inline Elements

A *Flowable* is a document element that is placed on a page. It is usually a part of a document tree. Flowables at one level in a document tree are rendered one below the other.

Here is schematic representation of an example document tree:

```
|- Section
|   |- Paragraph
|   \- Paragraph
\- Section
    |- Paragraph
    |- List
    |   |- ListItem
    |   |   |- Paragraph (item label; a number or bullet symbol)
    |   |   \- StaticGroupedFlowables (item body)
    |   |       \- Paragraph
    |   \- ListItem
    |       \- Paragraph
    |   |   \- StaticGroupedFlowables
    |   |       \- List
    |   |           |- ListItem
    |   |           |   \- ...
    |   |           \- ...
    \- Paragraph
```

This represents a document consisting of two sections. The first section contains two paragraphs. The second section contains a paragraph followed by a list and another paragraph. All of the elements in this tree are instances of *Flowable* subclasses.

`Section` and `List` are subclasses of *GroupedFlowables*; they group a number of flowables. In the case of *List*, these are always of the `ListItem` type. Each list item contains an item number (ordered list) or a bullet symbol (unordered list) and an item body. For simple lists, the item body is typically a single *Paragraph*. The second list item contains a nested *List*.

A *Paragraph* does not have any *Flowable* children. It is however the root node of a tree of *inline elements*. This is an example paragraph in which several text styles are combined:

```
Paragraph
 |- SingleStyledText('Text with ')
 |- MixedStyledText(style='emphasis')
 |    |- SingleStyledText('multiple ')
 |    \- MixedStyledText(style='strong')
 |         |- SingleStyledText('nested ')
 |         \- SingleStyledText('styles', style='small caps')
 \- SingleStyledText('.')
```

The visual representation of the words in this paragraph is determined by the applied style sheet. Read more about how this works in the next section.

Besides *SingleStyledText* and *MixedStyledText* elements (subclasses of *StyledText*), paragraphs can also contain InlineFlowables. Currently, the only inline flowable is InlineImage.

The common superclass for flowable and inline elements is *Styled*, which indicates that these elements can be styled using the style sheets which are discussed next.

7.2 Styling Document Elements

Rinohtype's style sheets are heavily inspired by CSS, but add some functionality that CSS lacks. Similar to CSS, rinohtype makes use of so-called *selectors* to select document elements (flowables and inline elements) to style.

Unlike CSS however, these selectors are not directly specified in a style sheet. Instead, all selectors are collected in a *matcher* where they are mapped to descriptive labels for the selected elements. A *style sheets* assigns style properties to these labels. Besides the usefulness of having these labels instead of the more cryptic selectors, a matcher can be reused by multiple style sheets, avoiding duplication.

7.2.1 Selectors

Selectors in rinohtype select elements of a particular type. The *class* of a document element serves as a selector for all instances of the class (and its subclasses). The *Paragraph* class is a selector that matches all paragraphs in the document, for example:

```
Paragraph
```

As with CSS selectors, elements can also be matched based on their context. For example, the following matches any paragraph that is a direct child of a list item (the list item label):

```
ListItem / Paragraph
```

Python's ellipsis can be used to match any number of levels of elements in the document tree. The following selector matches paragraphs at any level inside a table cell:

```
TableCell / ... / Paragraph
```

Selectors can select all instances of *Styled* subclasses. These include *Flowable* and *StyledText*, but also TableSection, TableRow, Line and Shape. Elements of some of the latter classes only appear as children of other flowables (such as Table).

Similar to a HTML element's *class* attribute, *Styled* elements can have an optional *style* attribute which can be used when constructing a selector. This one selects all styled text elements with the *emphasis* style, for example:

```
StyledText.like('emphasis')
```

The `Styled.like()` method can also match **arbitrary attributes** of elements by passing them as keyword arguments. This can be used to do more advanced things such as selecting the background objects on all odd rows of a table, limited to the cells not spanning multiple rows:

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

The argument passed as *row_index* is a slice object that is used for extended indexing[2]. To make this work, `TableCell.row_index` is an object with a custom `__eq__()` that allows comparison to a slice.

Rinohtype borrows CSS's concept of specificity to determine the "winning" selector when multiple selectors match a given document element. Each part of a selector adds to the specificity of a selector. Roughly stated, the more specific selector will win. For example:

```
ListItem / Paragraph                    # specificity (0, 0, 0, 0, 2)
```

wins over:

```
Paragraph                               # specificity (0, 0, 0, 0, 1)
```

since it matches two elements instead of just one.

Specificity is represented as a 5-tuple. The last four elements represent the number of *location* (currently not used), *style*, *attribute* and *class* matches. Here are some selectors along with their specificity:

```
StyledText.like('emphasis')             # specificity (0, 0, 1, 0, 1)
TableCell / ... / Paragraph             # specificity (0, 0, 0, 0, 2)
TableCell.like(row_index=2, rowspan=1)  # specificity (0, 0, 0, 2, 1)
```

Specificity ordering is the same as tuple ordering, so (0, 0, 1, 0, 0) wins over (0, 0, 0, 5, 0) and (0, 0, 0, 0, 3) for example. Only when the number of style matches are equal, the attributes match count is compared and so on.

In practice, the class match count is dependent on the element being matched. If the class of the element exactly matches the selector, the right-most specificity value is increased by 2. If the element's class is a subclass of the selector, it is only increased by 1.

The first element of the specificity tuple is the *priority* of the selector. For most selectors, the priority will have the default value of 0. The priority of a selector only needs to be set in some cases. For example, we want the `CodeBlock` selector to match a `CodeBlock` instance. However, because `CodeBlock` is a *`Paragraph`* subclass, another selector with a higher specificity will also match it:

```
CodeBlock                               # specificity (0, 0, 0, 0, 2)
DefinitionList / Definition / Paragraph # specificity (0, 0, 0, 0, 3)
```

To make sure the `CodeBlock` selector wins, we increase the priority of the `CodeBlock` selector by prepending it with a + sign:

```
+CodeBlock                              # specificity (1, 0, 0, 0, 2)
```

In general, you can use multiple + or − signs to adjust the priority:

```
++CodeBlock                             # specificity (2, 0, 0, 0, 2)
---CodeBlock                            # specificity (-3, 0, 0, 0, 2)
```

7.2.2 Matchers

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

2 Indexing a list like this `lst[slice(0, None, 2)]` is equivalent to `lst[0::2]`.

```
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
```

7.2.3 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* in the *Quickstart* guide.

A style sheet file contains a number of sections, denoted by a section title enclosed in squate 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.

Note:

The supported attributes and format of attribute values have not not yet been fully documented. Please look at the included style sheets for now.

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.

Custom Selectors

There is limited support for defining new selectors directly in a style sheet file. This allows making tweaks to an existing style sheet without having to create a new *StyledMatcher*, but should be used sparingly.

The syntax for specifying a selector for a style is the same as for element class, with the additional requirement that the element type name needs to be followed by parentheses. Arguments to be passed to `Styled.like()` can be included within the parentheses. For example:

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

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

Variables

Variables can be used for values that are used in multiple style definitions. This example declares a number of typefaces to allow easily replacing the fonts in a style sheet:

```
[VARIABLES]
mono_typeface=TeX Gyre Cursor
serif_typeface=TeX Gyre Pagella
sans_typeface=Tex Gyre Heros
thin_black_stroke=0.5pt,#000
blue=#20435c
```

It also defines the *thin_black_stroke* line style for use in table and frame styles, and a specific color labelled *blue*. These variables can be referenced in style definitions as follows:

```
[code block]
typeface=$(mono_typeface)
font_size=9pt
text_align=LEFT
indent_first=0
space_above=6pt
space_below=4pt
border=$(thin_black_stroke)
padding_left=5pt
padding_top=1pt
padding_bottom=3pt
```

Another stylesheet can inherit (see below) from this one and easily replace fonts in the document by overriding the variables.

Style Attribute Resolution

The style system makes a distinction between text (inline) elements and flowables with respect to how attribute values are resolved.

Text elements by default inherit the properties from their parent. Take for example the *emphasis* style definition from the example above. The value for style properties other than *font_slant* (which is defined in the *emphasis* style itself) will be looked up in the style definition corresponding to the parent element. That can be another *StyledText* instance, or a *Paragraph*. If that style definition neither defines the style attribute, the lookup proceeds recursively, moving up in the document tree.

For **flowables**, there is no fall-back to the parent's style by default. A base style can be explicitly specified however. If a style attribute is not present in a particular style definition, it is looked up in the base style.

This can also help avoid duplication of style information and the resulting maintenance difficulties. In the following example, the *unnumbered heading level 1* style inherits all properties from *heading level 1*, overriding only the *number_format* attribute:

```
[heading level 1]
typeface=$(sans_typeface)
font_weight=BOLD
font_size=16pt
font_color=$(blue)
line_spacing=SINGLE
space_above=18pt
space_below=12pt
number_format=NUMBER
label_suffix=' '

[unnumbered heading level 1]
base=heading level 1
number_format=None
```

When a value for a particular style attribute is set nowhere in the style definition lookup hierarchy its default value is returned. The default values for all style properties are defined in the class definition for each of the *Style* subclasses.

For text elements, it is possible to override the default behavior of falling back to the parent's style. Setting *base* to the label of a `TextStyle` or *ParagraphStyle* prevents fallback to the parent element's style.

7.2.4 Style Logs

When rendering a document, rinohtype will create a style log. It is written to disk using the same base name as the output file, but with a *.stylelog* extension. The information logged in the style log is invaluable when debugging your style sheet. It tells you which style maps to each element in the document.

The style log lists the document elements (as a tree) that have been rendered to each page, and for each element all matching styles are listed together with their specificity. No styles are listed when there aren't any selectors matching an element and the default values are used. The winning style is indicated with a `>` symbol. Styles that are not defined in the style sheet or its base(s) are marked with an `x`. If none of the styles are defined, rinohtype falls back to using the default style.

Here is an example excerpt from a style log:

```
...
  Paragraph('January 03, 2012', style='title page date')
       > (0,0,1,0,2) title page date
         (0,0,0,0,2) body
    SingleStyledText('January 03, 2012')
---------------------------------- page 3 ----------------------------------
#### ChainedContainer('column1')
  DocumentTree()
    Section(id='structural-elements')             demo.txt:62 <section>
         > (0,0,0,1,2) chapter
      Heading('1 Structural Elements')            demo.txt:62 <title>
           > (0,0,0,1,2) heading level 1
             (0,0,0,0,2) other heading levels
          MixedStyledText('1 Structural Elements')
            SingleStyledText('1')
            MixedStyledText(' ')
              SingleStyledText(' ')
            SingleStyledText('Structural Elements')
      Paragraph('A paragraph.')                   demo.txt:64 <paragraph>
           > (0,0,0,0,2) body
        MixedStyledText('A paragraph.')
          SingleStyledText('A paragraph.')
      List(style='bulleted')                      demo.txt:66 <bullet_list>
           > (0,0,1,0,2) bulleted list
        ListItem()
             x (0,0,1,0,4) bulleted list item
             > fallback to default style
          ListItemLabel('•')
               > (0,0,1,0,6) bulleted list item label
                 (0,0,0,0,2) list item label
              MixedStyledText('•')
                SingleStyledText('')
                SingleStyledText('•')
          StaticGroupedFlowables()                demo.txt:66 <list_item>
               > (0,0,0,0,3) list item body
...
```

CHAPTER 8

API Documentation

Note:

The API documentation is still incomplete.

8.1 Flowable

class rinoh.flowable.**Flowable** (*id=None*, *style=None*, *parent=None*)

A document element that can be "flowed" into a container on the page.
A flowable can adapt to the width of the container, or it can horizontally align itself in the container (see *HorizontallyAlignedFlowable*).

class rinoh.flowable.**FlowableStyle** (*base=None*, ***attributes*)

The Style for *Flowable* objects.

Parameters

- **space_above** (*DimensionBase*) -- Vertical space preceding the flowable. Default: 0
- **space_below** (*DimensionBase*) -- Vertical space following the flowable. Default: 0
- **margin_left** (*DimensionBase*) -- Left margin. Default: 0
- **margin_right** (*DimensionBase*) -- Right margin. Default: 0
- **padding_left** (*DimensionBase*) -- Left padding. Default: 0
- **padding_right** (*DimensionBase*) -- Right padding. Default: 0
- **padding_top** (*DimensionBase*) -- Top padding. Default: 0
- **padding_bottom** (*DimensionBase*) -- Bottom padding. Default: 0
- **keep_with_next** (*Bool*) -- Keep this flowable and the next on the same page. Default: False
- **border** (*Stroke*) -- Border surrounding the flowable. Default: None
- **border_left** (*Stroke*) -- Border left of the flowable. Default: None
- **border_right** (*Stroke*) -- Border right of the flowable. Default: None
- **border_top** (*Stroke*) -- Border above the flowable. Default: None
- **border_bottom** (*Stroke*) -- Border below the flowable. Default: None
- **background_color** (*Color*) -- Color of the area within the flowable's borders. Default: None

class rinoh.flowable.**FlowableState** (*flowable*, *_initial=True*)
Stores a flowable's rendering state, which can be copied.
This enables saving the rendering state at certain points in the rendering process, so rendering can later be resumed at those points, if needed.

8.1.1 No-Output Flowables

These flowables do not directly place anything on the page. All except *DummyFlowable* do have side-effects however. Some of these side-effects affect the rendering of the document in an indirect way.

class rinoh.flowable.**DummyFlowable** (*id=None*, *parent=None*)
A flowable that does not directly place anything on the page.
Subclasses can produce side-effects to affect the output in another way.

class rinoh.flowable.**AnchorFlowable** (*id=None*, *parent=None*)
A dummy flowable that registers a destination anchor.
Places a destination for the flowable's ID at the current cursor position.

class rinoh.flowable.**SetMetadataFlowable** (*parent=None*, ***metadata*)
A dummy flowable that stores metadata in the document.
The metadata is passed as keyword arguments. It will be available to other flowables during the rendering stage.

class rinoh.flowable.**WarnFlowable** (*message*, *parent=None*)
A dummy flowable that emits a warning during the rendering stage.

Parameters **message** (*str*) -- the warning message to emit

class rinoh.flowable.**PageBreak** (*id=None*, *style=None*, *parent=None*)
A flowable that optionally triggers a page break before rendering.
If this flowable's *page_break* style attribute is not None, it breaks to the page of the type indicated by *page_break* before starting rendering.

8.1.2 Labeled Flowables

class rinoh.flowable.**LabeledFlowable** (*label*, *flowable*, *id=None*, *style=None*, *parent=None*)
A flowable with a label.
The flowable and the label are rendered side-by-side. If the label exceeds the *label_max_width* style attribute value, the flowable is rendered below the label.

Parameters
- **label** (*Flowable*) -- the label for the flowable
- **flowable** (*Flowable*) -- the flowable to label

style_class
alias of *LabeledFlowableStyle*

class rinoh.flowable.**LabeledFlowableStyle** (*base=None*, ***attributes*)

Parameters
- **label_min_width** (*DimensionBase*) -- Minimum label width. Default: 12.00pt
- **label_max_width** (*DimensionBase*) -- Maximum label width. Default: 80.00pt
- **label_spacing** (*DimensionBase*) -- Spacing between a label andthe labeled

flowable. Default: 3.00pt

- **align_baselines** (*Bool*) -- Line up the baselines of the label and the labeled flowable. Default: True
- **wrap_label** (*bool*) -- Wrap the label at *label_max_width*. Default: False

Inherited parameters

- *FlowableStyle*: **space_above**, **space_below**, **margin_left**, **margin_right**, **padding_left**, **padding_right**, **padding_top**, **padding_bottom**, **keep_with_next**, **border**, **border_left**, **border_right**, **border_top**, **border_bottom**, **background_color**

class rinoh.flowable.**LabeledFlowableState** (*flowable*, *content_flowable_state*, *_initial=True*)

8.1.3 Grouping Flowables

class rinoh.flowable.**GroupedFlowables** (*id=None*, *style=None*, *parent=None*)

Groups a list of flowables and renders them one below the other.

Makes sure that a flowable for which *keep_with_next* is enabled is not seperated from the flowable that follows it.

Subclasses should implement *flowables()*.

style_class

alias of *GroupedFlowablesStyle*

flowables (*container*)

Generator yielding the *Flowable*s

class rinoh.flowable.**GroupedFlowablesStyle** (*base=None*, ***attributes*)

Parameters

- **title** (*StyledText*) -- Title to precede the flowables. Default: None
- **flowable_spacing** (*DimensionBase*) -- Spacing between flowables. Default: 0

Inherited parameters

- *FlowableStyle*: **space_above**, **space_below**, **margin_left**, **margin_right**, **padding_left**, **padding_right**, **padding_top**, **padding_bottom**, **keep_with_next**, **border**, **border_left**, **border_right**, **border_top**, **border_bottom**, **background_color**

class rinoh.flowable.**GroupedFlowablesState** (*groupedflowables*, *flowables*, *first_flowable_state=None*, *_initial=True*)

class rinoh.flowable.**StaticGroupedFlowables** (*flowables*, *id=None*, *style=None*, *parent=None*)

Groups a static list of flowables.

Parameters **flowables** (*iterable[Flowable]*) -- the flowables to group

class rinoh.flowable.**GroupedLabeledFlowables** (*id=None*, *style=None*, *parent=None*)

Groups a list of labeled flowables, lining them up.

8.1.4 Horizontally Aligned Flowables

class rinoh.flowable.**HorizontallyAlignedFlowable** (**args*, ***, *align=None*, *width=None*, ***kwargs*

)

A flowable with configurable width and horizontal alignment.
The *width* and *horizontal_align* control the width and horizontal alignment of the flowable.

style_class
alias of *HorizontallyAlignedFlowableStyle*

class rinoh.flowable.**HorizontallyAlignedFlowableStyle** (*base=None*, ***attributes*)

Parameters
- **width** (*DimensionBase*) -- The width of the flowable. Default: None
- **horizontal_align** (*HorizontalAlignment*) -- Horizontal alignment of the flowable. Default: left

Inherited parameters
- *FlowableStyle*: **space_above**, **space_below**, **margin_left**, **margin_right**, **padding_left**, **padding_right**, **padding_top**, **padding_bottom**, **keep_with_next**, **border**, **border_left**, **border_right**, **border_top**, **border_bottom**, **background_color**

class rinoh.flowable.**HorizontallyAlignedFlowableState** (*flowable*, *_initial=True*)

8.1.5 Floating Flowables

class rinoh.flowable.**Float** (*id=None*, *style=None*, *parent=None*)
A flowable that can optionally be placed elsewhere on the page.
If this flowable's *float* style attribute is set to `True`, it is not flowed in line with the surrounding flowables, but it is instead flowed into another container pointed to by the former's `Container.float_space` attribute.
This is typically used to place figures and tables at the top or bottom of a page, instead of in between paragraphs.

class rinoh.flowable.**FloatStyle** (*base=None*, ***attributes*)

Parameters **float** (*Bool*) -- Float the flowable to the top or bottom of the page. Default: True
Inherited parameters
- *FlowableStyle*: **space_above**, **space_below**, **margin_left**, **margin_right**, **padding_left**, **padding_right**, **padding_top**, **padding_bottom**, **keep_with_next**, **border**, **border_left**, **border_right**, **border_top**, **border_bottom**, **background_color**

8.2 Paragraph

class rinoh.paragraph.**ParagraphBase** (*id=None*, *style=None*, *parent=None*)
A paragraph of mixed-styled text that can be flowed into a `Container`.

style_class
alias of *ParagraphStyle*

render (*container*, *descender*, *state*, *first_line_only=False*)
Typeset the paragraph onto *container*, starting below the current cursor position of the container.

descender is the descender height of the preceeding line or *None*. When the end of the container is reached, the rendering state is preserved to continue setting the rest of the paragraph when this method is called with a new container.

class rinoh.paragraph.**Paragraph** (*text_or_items*, *id=None*, *style=None*, *parent=None*)

class rinoh.paragraph.**ParagraphStyle** (*base=None*, ***attributes*)

The Style for *Paragraph* objects

Parameters

- **indent_first** (*DimensionBase*) -- Indentation of the first line of text. Default: 0.00pt
- **line_spacing** (*LineSpacing*) -- Spacing between the baselines of two successive lines of text. Default: <rinoh.paragraph.DefaultSpacing object at 0x10fb7be48>
- **text_align** (*TextAlign*) -- Alignment of text to the margins. Default: justify
- **tab_stops** (*TabStopList*) -- List of tab positions. Default: []

Inherited parameters

- *FlowableStyle*: **space_above**, **space_below**, **margin_left**, **margin_right**, **padding_left**, **padding_right**, **padding_top**, **padding_bottom**, **keep_with_next**, **border**, **border_left**, **border_right**, **border_top**, **border_bottom**, **background_color**
- TextStyle: **typeface**, **font_weight**, **font_slant**, **font_width**, **font_size**, **font_color**, **font_variant**, **position**, **kerning**, **ligatures**, **hyphenate**, **hyphen_chars**, **hyphen_lang**

class rinoh.paragraph.**ParagraphState** (*paragraph*, *words*, *nested_flowable_state=None*, *_first_word=None*, *_initial=True*)

8.2.1 Inline Elements

class rinoh.text.**StyledText** (*id=None*, *style=None*, *parent=None*)

Base class for text that has a TextStyle associated with it.

style_class

alias of TextStyle

to_string (*flowable_target*)

Return the text content of this styled text.

is_script (*container*)

Returns *True* if this styled text is super/subscript.

script_level (*container*)

Nesting level of super/subscript.

height (*container*)

Font size after super/subscript size adjustment.

y_offset (*container*)

Vertical baseline offset (up is positive).

spans (*container*)
Generator yielding all spans in this styled text, one item at a time (used in typesetting).

class rinoh.text.**SingleStyledText** (*text*, *style=None*, *parent=None*)

class rinoh.text.**MixedStyledText** (*text_or_items*, *style=None*, *parent=None*)
Concatenation of *StyledText* objects.

append (*item*)
Append *item* (*StyledText* or str) to the end of this mixed-styled text.
The parent of *item* is set to this mixed-styled text.

8.2.2 Styling Properties

Line Spacing

class rinoh.paragraph.**LineSpacing**
Base class for line spacing types. Line spacing is defined as the distance between the baselines of two consecutive lines.

advance (*line*, *last_descender*, *container*)
Return the distance between the descender of the previous line and the baseline of the current line.

class rinoh.paragraph.**DefaultSpacing**
The default line spacing as specified by the font.

class rinoh.paragraph.**ProportionalSpacing** (*factor*)
Line spacing proportional to the line height.

class rinoh.paragraph.**FixedSpacing** (*pitch*, *minimum=<rinoh.paragraph.ProportionalSpacing object>*)
Fixed line spacing, with optional minimum spacing.

class rinoh.paragraph.**Leading** (*leading*)
Line spacing determined by the space in between two lines.

Tabulation

class rinoh.paragraph.**TabStop** (*position*, *align='left'*, *fill=None*)
Horizontal position for aligning text of successive lines.

get_position (*line_width*)
Return the absolute position of this tab stop.

8.2.3 Rendering Internals

class rinoh.paragraph.**GlyphAndWidth** (*glyph*, *width*)

class rinoh.paragraph.**GlyphsSpan** (*span*, *word_to_glyphs*, *chars=None*)

8.2.4 Miscellaneous Internals

class rinoh.paragraph.**HyphenatorStore**

8.3 Structure

class rinoh.structure.**List** (*flowables*, *id=None*, *style=None*, *parent=None*)

style_class
alias of *ListStyle*

class rinoh.structure.**ListStyle** (*base=None*, ***attributes*)

Parameters
- **ordered** (*Bool*) -- This list is ordered or unordered. Default: False
- **bullet** (*StyledText*) -- Bullet to use in unordered lists. Default: •

Inherited parameters
- *GroupedFlowablesStyle*: **title**, **flowable_spacing**
- *FlowableStyle*: **space_above**, **space_below**, **margin_left**, **margin_right**, **padding_left**, **padding_right**, **padding_top**, **padding_bottom**, **keep_with_next**, **border**, **border_left**, **border_right**, **border_top**, **border_bottom**, **background_color**
- NumberStyle: **number_format**
- LabelStyle: **label_prefix**, **label_suffix**, **custom_label**

8.4 Style

class rinoh.style.**Styled** (*id=None*, *style=None*, *parent=None*)
A document element who's style can be configured.

Parameters **style** (*str, Style*) -- the style to associate with this element. If *style* is a string, the corresponding style is lookup up in the document's style sheet by means of selectors.

style_class = None
The *Style* subclass that corresponds to this *Styled* subclass.

class rinoh.style.**Style** (*base=None*, ***attributes*)
Dictionary storing style attributes.
The style attributes associated with this *Style* are specified as class attributes of type Attribute.
Style attributes can also be accessed as object attributes.

8.4.1 Style Sheet

class rinoh.style.**StyleSheet** (*name*, *matcher=None*, *base=None*, *description=None*, *pygments_style=None*, ***user_options*)
Dictionary storing a collection of related styles by name.
*Style*s stored in a *StyleSheet* can refer to their base style by name.

Parameters
- **name** (*str*) -- a label for this style sheet
- **matcher** (*StyledMatcher*) -- the matcher providing the selectors the styles contained in this style sheet map to. If no matcher is given, the *base*'s matcher is used.

- **base** (*StyleSheet*) -- the style sheet to extend
- **description** (*str*) -- a short string describing this style sheet
- **pygments_style** (*str*) -- the Pygments style to use for styling code blocks

class rinoh.style.**StyleSheetFile** (*filename*, *matcher=None*, *base=None*, ***kwargs*)

Loads styles defined in a *.rts* file (INI format).

Parameters **filename** (*str*) -- the path to the style sheet file

StyleSheetFile takes the same optional arguments as *StyleSheet*. These can also be specified in the `[STYLESHEET]` section of the style sheet file. If an argument is specified in both places, the one passed as an argument overrides the one specified in the style sheet file.

class rinoh.style.**StyledMatcher** (*mapping_or_iterable=None*, ***kwargs*)

Dictionary mapping labels to selectors.

This matcher can be initialized in the same way as a `dict` by passing a mapping, an interable and/or keyword arguments.

8.5 Templates

The standard templates shipped with rinohtype all inherit from *DocumentTemplate*:

class rinoh.template.**DocumentTemplate** (*document_tree*, *strings=None*, *configuration=None*, *options=None*, *backend=None*)

parts

List of *DocumentPartTemplate*s that make up the document.

render (*filename_root=None*, *file=None*)

Render the document repeatedly until the output no longer changes due to cross-references that need some iterations to converge.

options_class

alias of `DocumentOptions`

Customization of the templates is performed by passing an instance of the template-specific *TemplateConfiguration* subclass as *configuration* on template instantiation.

class rinoh.template.**TemplateConfiguration** (*base=None*, ***kwargs*)

Parameters

- **stylesheet** (*StyleSheet*) -- The stylesheet to use for styling document elements. Default: StyleSheetFile(Sphinx)
- **paper_size** (*Paper*) -- The default paper size. Default: A4

8.5.1 Document Parts

The document part templates of which instances are include in *DocumentTemplate.parts* implement this interface:

class rinoh.template.**DocumentPartTemplate** (*name*, *right_page_template*, *left_page_template=None*, *page_number_format='number'*)

A template that produces a doument part, given a set of flowables, page templates and a page number format.

Parameters

- **name** (str) -- a descriptive name for this document part template
- **right_page_template** (*PageTemplateBase*) -- the page template for right pages
- **left_page_template** (*PageTemplateBase*) -- the page template for left pages. If not given, *right_page_template* is used.
- **page_number_format** (rinoh.number.NumberFormat) -- the number format used to style the page numbers in this document part. If it is different from the preceding part's number format, page numbering restarts at 1.

flowables (*document*)

Return a list of *rinoh.flowable.Flowable*s that make up the document part

The following document part templates are used in the standard document templates:

class rinoh.template.**TitlePartTemplate** (*name*, *right_page_template*, *left_page_template=None*, *page_number_format='number'*)

The title page of a document.

class rinoh.template.**ContentsPartTemplate** (*name*, *right_page_template*, *left_page_template=None*, *page_number_format='number'*)

The body of a document.

Renders all of the content present in the rinoh.document.DocumentTree passed to a *DocumentTemplate*.

class rinoh.template.**FixedDocumentPartTemplate** (*name*, *flowables*, *right_page_template*, *left_page_template=None*, *page_number_format='number'*)

A document part template that renders a fixed list of flowables.

Parameters **flowables** (list) -- a list of flowables to include in the document part

8.5.2 Page Templates

The document templates make use of page templates:

class rinoh.template.**PageTemplate** (*base=None*, ***attributes*)

Parameters

- **header_footer_distance** (*DimensionBase*) -- Distance of the header and footer to the content area. Default: 14.00pt
- **columns** (*Integer*) -- The number of columns for the body text. Default: 1
- **column_spacing** (*DimensionBase*) -- The spacing between columns. Default: 28.35pt
- **header_text** (*StyledText*) -- The text to place in the page header. Default: $SECTION_NUMBER(1) $SECTION_TITLE(1)
- **footer_text** (*StyledText*) -- The text to place in the page footer. Default: $PAGE_NUMBER/$NUMBER_OF_PAGES
- **chapter_header_text** (*StyledText*) -- The text to place in the header on a page that starts a new chapter. Default: None
- **chapter_footer_text** (*StyledText*) -- The text to place in the footer on a page that starts a new chapter. Default: None
- **chapter_title_flowables** (*Function*) -- Generator that yields the flowables to represent the chapter title. Default: <function chapter_title_flowables

at 0x10e68c840>

- **chapter_title_height** (*DimensionBase*) -- The height of the container holding the chapter title. Default: 150.00pt

Inherited parameters

- *PageTemplateBase*: **page_size**, **page_orientation**, **left_margin**, **right_margin**, **top_margin**, **bottom_margin**, **background**, **after_break_background**

class rinoh.template.**TitlePageTemplate** (*base=None*, ***attributes*)

Parameters

- **show_date** (*Bool*) -- Show or hide the document's date. Default: True
- **show_author** (*Bool*) -- Show or hide the document's author. Default: True
- **extra** (*StyledText*) -- Extra text to include on the title page below the title. Default: None

Inherited parameters

- *PageTemplateBase*: **page_size**, **page_orientation**, **left_margin**, **right_margin**, **top_margin**, **bottom_margin**, **background**, **after_break_background**

The base class for these collects the common options:

class rinoh.template.**PageTemplateBase** (*base=None*, ***attributes*)

Parameters

- **page_size** (*Paper*) -- The format of the pages in the document. Default: A4
- **page_orientation** (*PageOrientation*) -- The orientation of pages in the document. Default: portrait
- **left_margin** (*DimensionBase*) -- The margin size on the left of the page. Default: 85.04pt
- **right_margin** (*DimensionBase*) -- The margin size on the right of the page. Default: 85.04pt
- **top_margin** (*DimensionBase*) -- The margin size at the top of the page. Default: 85.04pt
- **bottom_margin** (*DimensionBase*) -- The margin size at the bottom of the page. Default: 85.04pt
- **background** (*BackgroundImage*) -- An image to place in the background of the page. Default: None
- **after_break_background** (*BackgroundImage*) -- An image to place in the background after a page break. Default: None

CHAPTER 9

Release History

9.1 Release 0.2.1 (2016-08-18)

New Features:

- optionally limit the width of large images and make use of this to simulate the Sphinx LaTeX builder behavior (#46)
- reStructuredText/Sphinx: support for images with hyperlinks (#49)
- record the styled page numbers in the PDF as page labels (#41)
- unsupported Python versions: prevent installation where possible (sdist) or exit on import (wheel)
- support Python 3.6

Bugfixes:

- make StyleSheet objects picklable so the Sphinx builder's rinoh_stylesheet option can actually be used
- Fix #47: ClassNotFound exception in Literal_Block.lexer_getter()
- Fix #45: Images that don't fit are still placed on the page
- don't warn about duplicate style matches that resolve to the same style

9.2 Release 0.2.0 (2016-08-10)

Styling:

- generate a style log (show matching styles) to help style sheet development
- keep_with_next style attribute: prevent splitting two flowables across pages
- stylesheets can be loaded from files in INI format
- check the type of attributes passed to styles
- source code highlighting using Pygments
- table of contents entries can be styled more freely
- allow hiding the section numbers of table of contents entries

- allow for custom chapter titles
- selectors can now also select based on document part/section
- various small tweaks to selectors and matchers
- various fixes relating to style sheets

Templates:

- configurable standard document templates: article and book
- a proper infrastructure for creating custom document templates
- support for left/right page templates
- make the Article template more configurable
- pages now have background, content and header/footer layers
- support for generating an index
- make certain strings configurable (for localization, for example)

Frontends:

- Sphinx: interpret the LaTeX configuration variables if the corresponding rinohtype variable is not set
- Sphinx: roughly match the LaTeX output (document template and style sheet)
- added a CommonMark frontend based on recommonmark
- added basic ePUB and DocBook frontends
- XML frontends: fix whitespace handling
- frontends now return generators yielding flowables (more flexible)

Command-line Renderer (rinoh):

- allow specifying a template and style sheet
- automatically install typefaces used in the style sheet from PyPI

Fonts:

- typefaces are discovered/loaded by entry point
- more complete support for OpenType fonts
- fix support for the 14 base Type 1 fonts

Images:

- more versatile image sizing: absolute width/height & scaling
- allow specifying the baseline for inline images
- several fixes in the JPEG reader

Miscellaneous:

- reorganize the Container class hierarchy
- fixes in footnote handling
- drop Python 3.2 support (3.3, 3.4 and 3.5 are supported)

9.3 Release 0.1.3 (2015-08-04)

- recover from the slow rendering speed caused by a bugfix in 0.1.2 (thanks to optimized element matching in the style sheets)
- other improvements and bugfixes related to style sheets

9.4 Release 0.1.2 (2015-07-31)

- much improved Sphinx support (we can now render the Sphinx documentation)
- more complete support for reStructuredText (docutils) elements
- various fixes related to footnote placement
- page break option when starting a new section
- fixes in handling of document sections and parts
- improvements to section/figure/table references
- native support for PNG and JPEG images (drops PIL/Pillow requirement, but adds PurePNG 0.1.1 requirement)
- new 'sphinx' stylesheet used by the Sphinx builder (~ Sphinx LaTeX style)
- restores Python 3.2 compatibility

9.5 Release 0.1.1 (2015-04-12)

First preview release

r

Symbols

A

B

C

D

E

F

S

T

W

Y