disseminate Documentation

Release 2.3.9

Justin L Lorieau

Jul 16, 2021

CONTENTS

Write documents once and publish for the web, in pdf, in epub and in other formats.

Disseminate is a document processing system to generate documents in multiple formats (html, pdf, epub etc.) suitable for academic and non-academic publishing. Disseminate can be used to publish textbooks, journal articles, articles, blogs and websites. It uses a simplified markup system as well as extensive and extensible tag functions.

CHAPTER

ONE

FEATURES: BASIC

1.1 1) Header and Body

Documents can optionally have a header, and they can optionally have a body.

```
---
title: This is my title
---
This is @i{my} body
```

The header is placed at the start of the document between `---` delimiters, and the body follows. The header follows a YAML format and the body follows the disseminate format.

1.2 2) Document Trees

Projects can be as simple as one document, or it can be a book comprising multiple parts and chapters. Documents are included in the header of the root document.

Listing 1: src/main.dm

```
---
title: Fundamentals of NMR Spectroscopy
author: Justin L Lorieau
include:
    part1/part1.dm
    part2/part2.dm
---
```

And each document can contain its own documents.

Listing 2: src/part1/part1.dm

```
---
part: Solution NMR
include:
    chapter1/chapter1.dm
    chapter2/chapter2.dm
---
```

Values set in the parent document (*e.g.* `author` carry over to the subdocuments)

1.3 3) Uniform Language

All tags use the same format. The following shows three valid tags.

```
@chapter
@chapter{My First Chapter}
@chapter[id=first-chapter]{My First Chapter}
```

All tags start with the `@` character and are followed by a keyword– `chapter` in this case. Any keyword is allowed, but only a subset have special functionality.

The **contents** of the tag are written between the '`{`' and '`}`' curly braces. The tag contents represent *what* will be rendered. The contents are optional and may be retrieved from the header values.

The **attributes** of the tag are written between the '`[`' and '`]`' square brackets. The tag attributes represent *how* a tag will be rendered. The attributes are also optional, and they can be key-value pairs (*e.g.* `[width=1in]` or positional (*e.g.* `[nolabel]`).

Additionally, tags can be arbitrarily nested, changing how the are rendered. For example, the following `@figure` tag includes an image and a caption.

```
@figure{
    @img[width=25%]{figures/fig1.svg}
    @caption{This is my first figure}
    }
```

1.4 4) Macros

Combinations of text and tags may be repeated frequently in a project. Macros can be used to simplify these blocks of text.

```
---
@H2O: H@sub{2}O
---
My favorite molecule is @H2O.
```

In this example, the `@H2O` macro is replaced with the text block `H@sub{2}O`.

1.5 5) Templates and Typography

A top priority for disseminate is the production of documents that follow good typographical style. This objective is important in improving the readability and aesthetics of documents.

First, the templates aim to follow good typographical rules. Templates are specified in the header of a document or in the header of a root document, if a project has multiple subdocuments.

```
---
templates: books/tufte
---
```

In this example, the final document is rendered using the Tufte book format.

Second, typographic elements are automicatically converted, such as opening and closing double quotes, en-dashes and em-dashes and ellipses.

1.6 6) Internal Labels

Labels to other documents, chapters, sections, figures and tables are handled consistently. Each of these content elements can have a label identifier specified.

```
@figure[id=intro-figure]{
    img{figures/fig1.png}
}
```

In this case, the label identifier is specified by the `id` attribute, and this figure's identifier is `intro-figure`.

References to labels are done with the `@ref` tag.

```
In @ref{intro-figure}, the introductory figure is shown.
```

The `@ref` tag accepts the label identifier, and it will automatically render the label in the format specified by the template.

Label identifiers must be unique for each document. If a non-unique identifier is used for 2 or more documents, then the document `doc_id` must be prepended with a semi-colon.

```
In @ref{chapters-chap1-dm:intro-figure}, the introductory figure is shown.
```

The `doc_id` is automatically generated by converting path delimiters (`/` and `\`) and periods (`.`) to dashes. The `doc_id` above was generated for the sub-document located at `chapters/chap1.dm`.

Alternatively, a unique `doc_id` can be specified in the header of a given document.

```
---
doc_id: introduction
---
```

1.7 7) Multiple Target Formats

Disseminate renders to multiple target formats. The target formats are specified in the header.

```
---
targets: html, tex, pdf, epub
---
```

In this case, the document will render in `html`, `tex`, `pdf` and `epub` formats.

1.8 8) Automatic Conversions

Disseminate includes builders to convert files to the formats needed for various targets, like html, tex, pdf and epub.

```
---
targets: html, tex, pdf
---
@img{media/figs/fig1.svg}
```

In the above example, the `fig1.svg` image is converted to `pdf` format when rendered in `tex` and `pdf` formats, and it is inserted directly for the `html` format.

When multiple conversion format options exist, image formats in vector graphic formats (`pdf`, `svg`) are favored.

1.9 9) Inline Plots and Diagrams

One of the major objectives for disseminate is to include raw data and methodology for rendering plots and diagrams, rather than insert these as simple images. Tags can convert plots and diagram input files or these can be inserted directly inline.

```
@asy{figures/diagram.asy}

@asy{
    size(200);
    draw(unitcircle);
}
```

In the above example, the first tag renders an asymptote diagram from the source file at `figures/diagram.asy`. The second tag uses the asymptote source code directly. These are converted to `svg` format for `html` and `epub` target formats, and they are converted to `pdf` format for `tex` and `pdf` target formats.

1.10 10) Equations

Equations are written in LaTeX format. For `tex` and `pdf` target formats, the equations are inserted directly, but for `html` and `epub` target formats, these are rendered as `svg` images.

```
This is my @eq{y = mx + b} linear equation.
```

Equations are rendered as vector graphi images in `html` and `epub`, instead of using a javascript library because the pages should render properly when javascript is disabled, the libraries only support a subset of LaTeX's math functionality, and javascript might not render properly in some target formats, like `epub`. Rendered equations have access to an extended set of LaTeX packages, including `amsmath` and `mathtools`.

CHAPTER

TWO

FEATURES: ADVANCED

2.1 1) Document Inheritance

Document headers define how a document and its sub-documents are rendered to target formats. Document headers are loaded into a context, which is a special `dict` in python. Most header values are copied to sub-documents, which may be overriden.

Consider the following project document tree.

```
src/main.dm
src/chapters/chap1.dm
src/chapters/chap2.dm
```

The project's `title` and `author` could be specified in the root document.

Listing 1: src/main.dm

```
---
title: This is my book
author: Debra Danson
template: books/novel
include:
    chapters/chap1.dm
    chapters/chap2.dm
---
@titlepage
```

The sub-documents carry the values from the parent document.

Listing 2: src/chapters/chap1.dm

```
---
chapter: My first chapter
---
@chapter
@author
```

In this case, the author field from `src/main.dm` will be inserted in `src/chapters/chap.dm`.

2.2 2) Target Attributes

Tag attributes generally apply to all document target formats. However, sometimes attributes need to be specified for specific document targets. For example, this feature can be helpful in customizing the position of images when the document is rendered on a page with the `pdf` document target.

Target attributes are specified by ending an attribute with a period and the document target.

```
@img[width=80% width.html=40%]{figures/fig1.png}
```

In this case, the image's width will be 80% of the text width for all document targets except `.html`, which will render the image's width to 40% of the text width.

Target attributes also apply to positional arguments.

```
@marginfig[id=perfect-vs-ideal-gas-3d -20em.tex]{
    @img[width=100%]{figures/fig1.1.png}
    @caption{Pressure vs. volume and temperature for @CO2 gas
       demonstrates that the real gas pressure is less than the
       perfect gas pressure.}
}
```

In this example, the `-20em.tex` positional argument offsets the margin figure's vertical position.

2.3 3) Label Formats

Templates may adopt different formats for the presentation of label references from `@ref` tags. These can be customized in the header of the root document or any of the sub-documents with the `label_fmts` entry.

```
---
label_fmts:
    heading: "@label.tree_number. @label.title"
    heading_title: "@label.title"
    heading_part: "Part @label.part_number. @label.title"
    heading_chapter: "Chapter @label.chapter_number. @label.title"
    caption_figure: "@b{Fig. @label.number}"
---
```

Label formats are written in disseminate format and, optionally, have access to the `@label` metadata.

For headings in general, they will adopt the most specific label format. In this case, `heading_title` takes precedence over `heading` for `@title` labels, so all titles will simply present the label's title. However, a `@section` heading will use the `heading` label format, since a `header_section` label format was not specified.

Additionally, target-specific label formats can be presented.

```
---
label_fmts:
    heading_title: "Title: @label.title"
    heading_title_html: "My HTML Title: @label.title"
---
```

In this example, the `heading_title_html` label format will be used to format references to `@title` headings with `html` targets whereas the `heading_title` label format will be used for all other targets.

Finally, templates may reset the heading counters for different types of headings with the `label_resets` header entry.

```
---
label_resets:
    part: chapter, section, subsection, subsubsection
    chapter: section, subsection, subsubsection, figure, table
---
```

In the above example, the number (counter) for the chapter, section, subsection and subsubsection are reset every time there is a new `@part` is encountered. Likewise, the section, subsection, subsubsection, figure and table numbers are reset every time a new `@chapter` is encountered.

2.4 4) Signal Processing

Disseminate integrates a customized signal dispatch system. The system is designed similarly to Blinker, but it allows receivers to register an order for processing in sequence.

The processing of documents, tags, labels, building is initiated by a signal dispatch and handled in a factory pattern with signal receivers. This system allows new functionality to be easily added, like plug-ins, while decoupling the various sub-modules of disseminate.

The listing of current signals and receivers can be accessed from the *commandline interface*.

2.5 5) Intelligent Building

In many respects, disseminate is a software construction tool like SCons or GNU make: document source files (`.dm`) are parsed and converted to tags, and external tools may optionally build some of the dependencies, like images, plots and diagrams.

The disseminate build system includes a number of features to quickly and correctly build the target documents:

1. Each document target has its own builder, which comprises of a nested tree of sequential and parallel builders.
2. Builders are selected as needed from the specified document target–*e.g.* a `pdf2svg` builder is added for `svg` images when the document is rendered to the `html` document target, but the `pdf` is only added (copied) for `tex` and `pdf` document targets.
3. The build system uses a multithreaded and multiprocessing implementation to build documents in parallel to the extent possible. This parallelization reduces overall build times from 10s of minutes to 10s of seconds.
4. Intermediate files are cached in a `.cache` subdirectory.

5. Like SCons, build decisions are made based on the hash of files rather than modification times (like GNU make)
6. Additional dependencies like labels hashes are included in the build. References to labels that have changed from other documents will trigger a new build.
7. The build system is designed to operate transparently to the user.

2.6 6) Webserver Preview

The CLI includes a webserver based on Sanic to present the rendered documents with the `html` document target.

The webserver previewer is started from the command line.

CHAPTER

THREE

QUICKSTART

Follow the installation instructions to get disseminate working.

3.1 Terminal

3.1.1 Step 1: Setup a project

A disseminate project is a collection of documents that come together to render into textbooks, books, articles or essays.

The simplest project consists of a single document, typically in its own directory.

3.1.2 Step 2: Create a first document

In your favorite text editor, edit a first document (`src/main.dm`).

```
---
title: My first document
author: John A. Doe
targets: html
---
@chapter{This is my first chapter}

I am @i{here} to show you how this works.
```

3.1.3 Step 3: Build the document

Use disseminate to build the document.

3.1.4 Step 4: View the document

Use the disseminate previewer to view the document website.

Use a web browser to visit the webserver on your computer at **http://127.0.0.1:8899**. This will present the document index for your project.

Document Listing.svg

Clicking on the `html` link will show the html rendered version of your document.

first document.svg

CHAPTER

FOUR

MACOS (HOMEBREW)

Homebrew is a package manager for MacOS.

Install the Disseminate homebrew package.

4.1 TeX Live

(Optional) For building PDFs, a LaTeX installation is needed. The `basictex` package can be installed with Homebrew. The needed packages can be installed on a TeXLive distribution.

CHAPTER

FIVE

MACOS (PIP)

5.1 Pip

Disseminate is installed using pip.

5.2 TeX Live

(Optional) Rendering PDFs requires a MacTeX installation.

The needed packages can be installed on a TeXLive distribution.

CHAPTER

SIX

LINUX (PIP)

6.1 Pip

Disseminate is installed using pip.

6.2 TeX Live

(Optional) Rendering PDFs requires a LaTeX installation.

The needed packages can be installed on a TeXLive distribution.

CHAPTER

SEVEN

HEADER

Headers contain useful meta information on the document and sub-documents, which may impact how the document, sub-documents and tags are rendered.

Headers are optionally present at the start of a disseminate document. Headers are enclosed by 3 or more hyphens ('-').

```
---
title: Disseminate Headers
author: Justin L Lorieau
template: component
targets: html, tex, pdf
include:
  part1/introduction.dm
  part1/cakes.dm
  part1/tarts.dm
---
```

Since documents can include sub-documents, the context of a parent document is copied over to the sub-documents, which in turn, can overwrite these values.

7.1 Entries

7.1.1 Basic Entries

title The project's title

examples

```
title: My First Document
```

short The document's *short* title

examples

```
short: First
```

author The document's author

examples

```
author: Justin L Lorieau
```

targets The document target formats

values `html`, `tex`, `pdf`, `epub`, `txt`

examples

```
targets: html, tex, pdf
```

template The template to use in rendering the document

values

- `articles/basic`
- `books/novel`
- `books/tufte`
- `default`
- `reports/basic`

examples

```
template: book/tufte
```

include The sub-documents to include in a project

notes See the section on *Includes*.

7.1.2 Render Options

doc_id The unique document identifier, which may be used to reference labels in other documents.

examples

```
doc_id: chapters-chap1-dm
```

label_fmts The formats for labels in rendering the document.

examples

```
label_fmts:
   document: "@label.title"
   ref_document: "@label.title"
   heading: "@label.tree_number."
   heading_part: "Part @label.part_number. "
   heading_chapter': "Chapter @label.chapter_number. "
```

label_resets Specify which label kinds reset the counters f9r other label kinds.

examples A project may need to have the the chapter, section and subsection numbers reset every time there is a new part. To achieve this sort of reset, the following entry would be entered in the heading of the root document.

```
label_resets:
   part: chapter, section, subsection
```

relative_links If True (default), html links are referenced relative to the document's path. If False, links are absolute.

examples

```
relative_links: True
```

`base_url`

If absolute links are used and `relative_links` is False, then the base_url string will be prepended to links.

examples

```
base_url: /{target}/{subpath}
```

`process_paragraphs` A list of context entries for which paragraphs should be processed

notes By default, automated paragraph processing is enabled for the body entry.

examples

```
process_paragraphs: body
```

7.2 Includes

Sub-documents can be included in the header. Sub-documents may represent parts of books, chapters or sections. Sub-documents can be furthermore nested to form projects with multiple sub-levels. The include statement lists the sub-documents directly subordinate to a document.

`include` Sub-documents to include in the document tree

notes Sub-documents and their paths are listed one per line with at least two spaces before each entry. The paths are relative to the current document.

examples

```
include:
  part1/introduction.dm
  part1/cakes.dm
  part1/tarts.dm
```

A **project** consists of one root document and, possibly, one more included documents. It is alternatively known as a document tree.

A **root document** is the first document in a project, and it is not included by any other document.

A **child document** is a document that was included by a document.

A **parent document** is the document that was included by a document.

7.3 Inheritance

Most header entries from parent documents are available to child documents. If child documents specify their own version of these entries, then these values will be overwritten.

For example, a parent document might specify a `title` entry in the header. If the parent document includes a child document, then the `title` entry will be the same for the child document, unless the child document's header also specifies a `title` entry, in which case this entry will be used.

For this reason, it is useful to specify settings for an entire project in the header of the root document, then to overwrite these values (if needed) in the child document headers.

From a technical perspective, header entries are loaded into the context for a document. The context is a Python dictionary that holds the variables needed to render a document.

7.4 Additional Notes

7.4.1 How do I include quotations in header entries?

Quotations within a sentence or string will not be removed

example

```
title: This is my "test" title
```

However, quotations on the end of a sentence or string will be removed. To include these quotations, simply add the quotation mark twice on each side of the entry.

example

```
title: ""This is my test title""
```

In the above example, the title will be `"This is my test title"` with the double quotations.

CHAPTER

EIGHT

TAGS

Tags can add meaning to a block of text. In disseminate, all tags are allowed. Tags start with an `@` character, followed by a letter and one ore more letters or numbers. Tags may have contents, which are contained within curly braces. The following list shows valid tags:

```
@example{...}
@h1{...}
@abhängigkeit{...}
@friendship
```

Attributes

Additionally, tags may contain attributes. Attributes are added with square brackets before the contents.

```
@h1[id="Intro"]{Introduction}
@asy[scale=1.0]{dot((20,0));}
@eq[env=alignat* 2]{y &= x}
```

Tag contents dictate *what* to present, and tag attributes dictate *how* the contents are presented.

Target-specific Attributes. Attributes for specific targets can be passed to the processor by appending the target's name to the attribute:

```
@img[width.tex=300 width.html=150 id=figure-1]{...}
```

The customization of specific targets is typically reserved for the final stages of publication so that the targets have the final desired appearance.

8.1 Preamble

The preamble tags format introductory material for a document or project.

`@titlepage` Render a title page for the document

Examples

```
@titlepage{}
```

`@toc{...}` Render a table of contents. The table of contents behaves as a site map for the project

contents `all`

Include labels from all sub-documents in the TOC. If all is not specified, only the labels for the current document will be presented.

`documents`

Render a document TOC with the document and heading entries.

`headings`

Render a TOC with heading entries.

`figure`

Render a TOC with figure caption entries.

content modifiers `collapsed` (default)

Show only the documents without headings. Pertains to `all documents` and `all headings`.

`expanded`

Show all documents are including all headings. Pertains to `all documents` and `all headings`.

`abbreviated`

show all documents but only show headings for the current document. Pertains to `all documents` and `all headings`.

attributes `header`

Include a 'Table of Contents' heading for the TOC. Note that this heading won't appear in the TOC.

examples

```
@toc[header]{all documents}
@toc{all documents collapsed}
```

8.2 Headings

Headings are used to group and demarcate parts of a document or project.

`@part{...}` A part heading

aliases `@h1`

attributes `id=x`

The heading's marker label

`nolabel`

If specified, a label is not created with this heading for TOC entries and links

`short=x`

The short title, which can be used in the TOC

`fmt=x`

The format for a tag's label

examples

```
@chapter{Object Decorators}
@chapter[fmt="Chapter {label.tree_number}. '{label.short}'"]
↪{Introduction}
```

@chapter{...} A chapter heading

aliases `@h2`

attributes `id=x`

The heading's marker label

`nolabel`

If specified, a label is not created with this heading for TOC entries and links

`short=x`

The short title, which can be used in the TOC

`fmt=x`

The format for a tag's label

examples

```
@chapter{Object Decorators}
@chapter[fmt="Chapter {label.tree_number}. '{label.short}'"]
↪{Introduction}
```

@section{...} A section heading

aliases `@h3`

attributes `id=x`

The heading's marker label

`nolabel`

If specified, a label is not created with this heading for TOC entries and links

`short=x`

The short title, which can be used in the TOC

`fmt=x`

The format for a tag's label

examples

```
@section{Introduction}
@section[id="chapter1-introduction"]{Introduction}
@h2{Introduction}
```

@subsection{...} A subsection heading

aliases `@h4`

attributes `id=x`

The heading's marker label

`nolabel`

If specified, a label is not created with this heading for TOC entries and links

`short=x`

The short title, which can be used in the TOC

`fmt=x`

The format for a tag's label

examples

```
@subsection{Methods}
@h3{Methods}
```

`@subsubsection{...}` A subsubsection heading

aliases `@h5`

attributes `id=x`

The heading's marker label

`nolabel`

If specified, a label is not created with this heading for TOC entries and links

`short=x`

The short title, which can be used in the TOC

`fmt=x`

The format for a tag's label

examples

```
@subsubsection{Titration Procedure}
@h4{Titration Procedure}
```

`@paragraph{...}` A paragraph heading

aliases `@h6`

attributes `id=x`

The paragraph's marker label

html In html, this tag will be rendered as a `<span class="paragraph-heading">` instead of an `<h5>` element.

note This tag is distinct from the `@p`, which is used to identify a paragraph element.

examples

```
@paragraph{Group A}. The first group ...
@h5{Group A}. The first group ...
```

8.2.1 Identifiers and Labels

By default, all headings have a *unique* identifier and label. Labels allow other portions of a project to reference the heading.

If a heading without a label is desired, the `nolabel` attribute can be used. Headings without a label cannot be linked and referenced in the project, and the heading will not be included in Tables of Content.

Otherwise, it is recommended to use an identifier. An identifier is specified with the `id=x` attribute, and it should be *unique* for the project. If an identifier is not specified, an identifier will be generated.

8.2.2 Empty Contents

If the contents are empty, the tag will search the header entries to see if an entry with the same name is present. For example, if the header has an entry `chapter: My First Chapter`, then inserting the `@chapter` tag in the body will use the 'My First Chapter' text as its contents.

8.3 Text Formatting

Text formatting tags are used to emphasize text in different ways and to introduce special characters.

`@bold{...}` Emphasize text by with bold

aliases `@b` `@textbf`

examples

```
@bold{This text is bold!}
@b{This text is bold!}
@textbf{This text is bold!}
```

`@italics{...}` Emphasize text by with italics

aliases `@i` `@textit`

examples

```
@italics{This text is in italics}
@i{This text is in italics}
@textit{This text is in italics}
```

`@sup{...}` Superscript text

examples

```
@sup{1}H
```

`@sub{...}` Subscript text

examples

```
H@sub{2}O
```

`@supsub{...}` A superscript followed by a subscript text. This tag formats the superscript directly above the subscript

content The superscript and the subscript are separated by two ampersands (&&)

examples

```
@supsub{12 && 6}C
```

@symbol{...} Add a symbol

aliases `@smb`

examples

```
@symbol{alpha}-helix
```

@verb{...} Mark text as verbatim–i.e. do not process the text and present the text without modification.

tex In tex, this tag will be rendered as an inline `\verb||` command.

examples

```
My @v{@bold{bold}} tag.
```

verbatim{...} Mark a *block of text* as verbatim–*i.e.* do not process the text and present the text without modification.

tex In tex, this tag will be rendered as a `\begin{verbatim}...\end{verbatime}` environment.

examples

```
@verbatim{My verbatim text}
```

8.4 References

Reference tags create links to other documents and content like figures and tables.

@ref{...} A reference to a label

content The unique label identifer (`label_id`) for the content element. The `label_id` must be unique for each document. If a `label_id` is reused between multiple documents, then the `doc_id` must also be specified with 2 colons as separator. *i.e.* `doc_id::label_id`. The `doc_id` can be specified in the *Header*.

examples

```
@ref{chap1::intro}
@ref{fig:overview}
```

8.5 Images

Tags to insert and format images

@img{...} Insert an image.

contents The path and filename of the image. The path can be either relative to the same directory as the document, the root directory of the project or and absolute path.

attributes `class=x`

the css class to use for the image in `html` targets.

`width=x%`

the width of the image using the percentage or px

(for all targets)

`height=x`

the height of the image using the specified number (for all targets)

target attributes `width.html=x`

the width of the image using the specified percentage or px. The width is used in the `<img />` tag.

(`html` targets)

`height.html=x`

the height of the image using the specified number. The height is used in the `<img \>` tag.

(`html` targets)

`width.xhtml=x`

the width of the image using the specified percentage or px. The width is used in the `<img />` tag.

(`epub` and `xhtml` targets)

`height.xhtml=x`

the height of the image using the specified number. The height is used in the `<img \>` tag.

(`epub` and `xhtml` targets)

`width.tex=x`

the width of the image using the specified percentage or px. The width is used in the `\includegraphics` macro.

(`tex` and `pdf` targets)

`height.tex=x`

the height of the image using the specified number. The height is used in the `\includegraphics` macro.

(`tex` and `pdf` targets)

examples

```
@img[class=icon]{media/imgs/template.svg}
@img[tex.width=30% html.width=2.5in]{media/imgs/simulation.svg}
```

8.6 Equations

Tags to insert and format equations

@eq{...} Insert an equation

contents The equation in LaTeX format.

attributes `env=x`

The LaTeX environment to use in rendering the equation. By default, equations are rendered inline within paragraphs or as `align*` environments for block equations. Note that additional parameters may be needed, depending on the environment.

`color=x`

The color of the equation.

`bold`

Render the equation in bold

html For `html` targets, equations are rendered using LaTeX and inserted as `.svg` files.

tex/pdf For `tex` targets, equations are integrated directly in the `.tex` file.

epub/xhtml For `epub` and `xhtml` targets, equations are rendered using LaTeX and inserted as `.svg` files.

examples

```
@eq{y = x}
@eq[env=align]{y &= x}
@eq[env=alignat* 2]{y &= x}
```

8.7 Code Fragments

Code highlighting tags are used to represent source code in texts using pygments.

@code{...}

Highlight a code fragment

contents A code fragment or a path to a source file.

attributes `language`

The language to use in highlighting the source code fragment. See the pygments /docs/lexers/>`_ lexer listing.

examples

```
@code[python]{print('hello!')}
```

@dm{...}

Highlight *disseminate* code fragment

contents A code fragment or a path to a dm source file.

examples

```
@dm{@b{hello!')}
```

@python{...}

Highlight a *python* code fragment

contents A code fragment or a path to a python source file.

examples

```
@python{print('hello!')}
```

@html{...}

Highlight an *html* code fragment

contents A code fragment or a path to a html source file.

examples

```
@html{<b>hello!</b>}
```

@ruby{...}

Highlight a *ruby* code fragment

contents A code fragment or a path to a ruby source file.

examples

```
@ruby{print "hello!"}
```

@java{...}

Highlight a *java* code fragment

contents A code fragment or a path to a java source file.

examples

```
@java{System.out.println("hello!" );}
```

@javascript{...}

Highlight a *javascript* code fragment

contents A code fragment or a path to a javascript source file.

examples

```
@javascript{alert('Hello!');}
```

8.8 Asymptote Diagrams

Tags to insert and format asymptote vector graphics diagrams

@asy{...} Insert an asymptote image

contents The figure in Asymptote syntax or the location of an asymptote source file.

attributes `scale=x`

xcale the image by the specified factor

html For `html` targets, asymptote images are rendered using asymptote and inserted as `.svg` files.

tex For `tex` targets, asymptote images are inserted directly.

examples

```
@asy[scale=2.0]{
     size(200);
     draw(unitcircle);
     }
```

8.9 Figures

Tags to insert figures

@marginfig{...} Insert a figure in the margin

aliases `@marginfigure`

contents An `@img` tag and, optionally, a `@caption` tag

attributes `id=x`

The margin figure's marker label

`height_offset.tex`

(Tufte template) offset the margin figure by the given vertical dimension. (*ex:* `-15em.tex`)

examples

```
 @marginfig[id=my-graph]{
    @img{media/graph.svg}
    @caption{My first graph}
 }

@marginfig[-15em.tex]{
    @img{media/graph.svg}
    @caption{My second graph}
 }
```

@fig{...} Insert a figure in the main text.

aliases `@figure`

contents An `@img` tag and, optionally, a `@caption` tag

attributes `id=x`

The figure's marker label

examples

```
@fig[id=my-graph]{
   @img{media/graph.svg}
   @caption{My first graph}
     }
```

`@fullfig{...}` Insert a full figure that spans the main text and margin.

aliases `@fullfigure`, `@ffig`

contents An `@img` tag and, optionally, a `@caption` tag

attributes `id=x`

The figure's marker label

examples

```
@fullfig[id=my-graph]{
   @img{media/graph.svg}
   @caption{My first graph}
     }
```

`@panel{...}` Insert a panel in a figure.

contents An `@img` tag and text

attributes `width="x"`

(Required) The panel's width in percentage, px units

`width.html="x"`

The panel's width in percentage, px units (for `html` targets)

`width.tex="x"`

The panel's width in percentage, px units (for `tex` targets)

examples

```
@panel[width=30%]{
   @img{media/graph.svg}
     }
```

8.10 Captions

Tags to insert captions for figures and tables

`@caption{...}` Insert a caption

contents The figure caption

attributes `id=x`

The caption marker label

notes The `@caption` tag is designed to be used with a figure tag or a table tag. A naked `@caption` is allowed, but a label will not be registered for it.

examples

```
@fig{@img{image.svg}
     @caption{My first figure}
     }
```

8.11 Table of Contents

Table of contents are used to create links to headings, content or other documents.

`@toc{...}` A table of contents

contents: label types `document`

List documents

`heading`

List headings like chapters, sections, subsections and subsubsections

`figure`

List figure captions

`table`

List tables

contents: modifiers `all`

List entries for all documents. By default, only the entries for the current document are listed.

`current`

List entries for the current document only (default)

`expanded`

When `all documents` or `all headings` is used, show all headers for all documents

`abbreviated`

When `all documents` or `all headings` is used, only show headers for the current document and document labels for the other documents

`collapsed`

When `all documents` or `all headings` is used, only show the document labels

examples

```
@toc{headings collapsed}
@toc{all headings expanded}
```

8.12 Navigation

Navigation tags are used to create html links to other documents, like the previous or next document.

@prev A link to the previous document

attributes `kind`

The kind of label to link to create the link to. By default, the link is created to the first heading label, like `@chapter`.

examples

```
@prev
```

@next A link to the next document

attributes `kind`

The kind of label to link to create the link to. By default, the link is created to the first heading label, like `@chapter`.

examples

```
@next
```

8.13 Feature Box

Feature boxes are text boxes that present additional, tangential information to the text. Feature boxes are used to present examples, tips or background on a subject.

@featurebox{...} A general feature box

examples

```
@featurebox{
   This is a feature box.

   It has 2 paragraphs
}
```

@examplebox{...} A feature box for presenting examples

examples

```
@examplebox{
   @b{Problem:} This is an example box. How is it used?

   @b{Answer:} This is how.
}
```

@problembox{...} A feature box for presenting problem questions

examples

```
@problembox{
   @b{Problem:} This is an example box. How is it used?
}
```

8.14 Tables

Tags to insert and format tables

@table{...} Insert a table

contents A data tag and, optionally, a caption tag.

attributes `id=x`

The identifier for the table caption

`class=x`

The formatting class for the table

examples

```
@table{
   @csv{
     First Name, Last Name
     John, Smith
     Betty, Sue
     Derek, Johnson
   }
 }
```

The following table includes a csv file and a caption title.

```
@mtable{
   @caption{Populations by age group for tennis players}
   @csv{data/populations.csv}
 }
```

@margintable{...}

Insert a table in the margin

contents A data tag and, optionally, a caption tag.

attributes

`id=x`

The identifier for the table caption

`class=x`

The formatting class for the table

examples

```
@margintable{
   @caption{Populations by age group for tennis players}
   @csv{data/populations.csv}
 }
```

`@fulltable{...}`

Insert a table that spans the whole page

contents A data tag and, optionally, a caption tag.

attributes

`id=x`

The identifier for the table caption

`class=x`

The formatting class for the table

examples The following table includes a csv file and a caption title.

```
@fullable{
   @caption{Populations by age group for tennis players}
   @csv{data/populations.csv}
 }
```

8.15 Data

Tags to insert data. This tag is used in conjunction with other tags, like the `@table` tag.

`@csv{...}` Include comma-separated value (CSV) data.

contents Either a path for a file with CSV data or CSV data directly.

attributes `noheader`

The `csv` file does not include header labels in the first line.

examples The following example loads csv data from a file `data/class_histogram.csv` into a table.

```
@table{
  @csv{data/class_histogram.csv}
}
```

This example loads csv data directly into a table.

```
@table{
  @csv{
    First Name, Last Name
    John, Smith
    Betty, Sue
    Derek, Johnson
  }
}
```

This example loads csv data without a header line into a table.

```
@table{
  @csv[noheader]{
    John, Smith
    Betty, Sue
    Derek, Johnson
  }
}
```

CHAPTER

NINE

MACROS

Macros are used to simplify the entry of multiple tag commands.

9.1 Greek

The following macros add characters in Greek.

`@Alpha`

`@alpha`

`@Beta`

`@beta`

`@Gamma`

`@gamma`

`@Delta`

`@delta`

`@Epsilon`

`@epsilon`

`@Zeta`

`@zeta`

`@Eta`

`@eta`

`@Theta`

`@theta`

`@Iota`

`@iota`

`@Kappa`

`@kappa`

`@Lambda`

`@lambda`

`@Mu`

@mu

@Nu

@nu

@Xi

@xi

@Omicron

@omicron

@Pi

@pi

@Rho

@rho

@Sigma

@sigma

@Tau

@tau

@Upsilon

@upsilon

@Phi

@phi

@Chi

@chi

@Psi

@psi

@Omega

@omega

9.2 Science

9.2.1 Units

@deg

@degC C

9.3 Chemistry

9.3.1 Isotopes

`@1H` ^{1}H

`@13C` ^{13}C

`@15N` ^{15}N

`@19F` ^{19}F

`@31P` ^{31}P

9.3.2 Chemicals

`@H2O` H_2O

CHAPTER

TEN

DOCUMENTS

Each disseminate source file is a document. Documents are rendered into target formats.

10.1 Trees

A document may have sub-documents which, in turn, may each have sub-documents. The tree of documents forms a *project*, and the top-level document is the *root document*.

Document trees are created by *including* sub-documents in a document with the *include* entry in a document's header.

Projects are organized in a source directory. By default, these are placed in a `src` directory. One or more disseminate source files and dependent media files are placed in the root or sub-directories `src` directory.

10.2 Templates

Documents use *Templates* to render documents. Disseminate comes with a series of built-in templates, or custom, user-defined templates can be placed in the `src` directory and specified in the `template` directory.

10.3 Path Precedence

Documents and projects may include image, media and data files. Paths are stored in a list in the `'path'` entry of the document context. Paths are searched in the following order:

1. *Template paths*. Includes template files like `.css` stylesheets. May include the template path for derived templates followed by the parent template path. Defined in *document.receivers.process_headers()*.
2. *Local path*. The path local to the document. For example, the file `src/chapter1/figures/fig1.png` included in `src/chapter1/chapter1.dm` will be used and copied to the target path `html/chapter1/figures/fig1.png`. Defined in *document.document_context.DocumentContext.reset()*.
3. *Project root path*. The path of the project root directory. For example, the file `src/media/figures/fig1.png` included in `src/chapter1/chapter1.dm` will be used and copied to the target path `html/media/figures/fig1.png`. Defined in *document.document_context.DocumentContext.reset()*.

CHAPTER

ELEVEN

TEMPLATES

Templates are used to render disseminate documents (`.dm`) to other formats, such as `.html`, `.tex` and `.pdf`. A variety of templates are built into disseminate.

11.1 Built-in Templates

Built-in or global templates reside within the disseminate project.

The following templates are available:

```
default
articles/basic
books/novel
books/tufte
reports/basic
```

The `default` template will be used unless another template is specified in the header of a document. The specified template will then be used for the document and all sub-documents.

CHAPTER

TWELVE

COMMAND-LINE

The command-line interface (CLI) is used to render disseminate documents and to preview rendered html documents with a built-in web server.

Disseminate is invoked on the command line through the `dm` command.

The disseminate main CLI includes the following help option:

Disseminate includes a series of sub-commands.

12.1 init

The `init` sub-command initializes and lists template projects to start a new project in disseminate.

The `init` help presents the following options.

The list of project starter templates can be listed.

The template names are listed in red. Detailed information on a project starter template can be listed with the `--info` flag. In the following example, the detailed information for the `books/tufte/textbook1` project starter template is listed.

A new project is initialized with the `init` command and the template name.

12.2 build

The `build` sub-command compiles the disseminate source to the target formats.

The `build` help presents the following optional arguments:

12.3 preview

The `preview` sub-command runs a web server on the local computer (localhost) to present the source and target format files of disseminate projects.

The web server is started as follows:

The `preview` sub-command help presents the following optional arguments:

12.4 setup

12.4.1 --check

Disseminate uses external software for various conversion and compilation tasks. The `--check` function reports whether these software dependencies are installed and available to disseminate.

12.4.2 --list-signals

Disseminate uses signals and receivers for modular and decoupled processing of documents, tags and other objects. The list of signals and attached receivers can be listed with the following command:

CHAPTER

THIRTEEN

API DOCUMENTATION

The Advanced Programming Interface (API) documentation lists the classes and functions for disseminate.

13.1 Attributes

Attributes are modifiers to tags that change *how* a tag is rendered. Attributes are represented by ordered dicts that can validate their entries.

exception disseminate.attributes.**AttributeFormatError**
Bases: Exception

An error was encountered in the format of attributes

class disseminate.attributes.**Attributes**(**args*, ***kwargs*)
Bases: dict

The attributes class is an ordered dict to manage and validate attribute entries.

Note: Attributes can either be key/value (ex: 'class=media') or positional (ex: 'red'). For positional arguments, the values are *PositionalValue* class objects.

---

append(*attr*, *value=<class 'disseminate.attributes.attributes._MissingAttribute'>*)
Set a value by appending to it, if it already exists, or creating it if it doesn't.

Parameters

attr [str] The attr (key) for the entry to set or append.

value [Optional[Any]] The value to set or append. If not specified, the attribute will be added as a positional argument

copy() → a shallow copy of D

filter(*attrs=None*, *target=None*, *sep='.'*, *sort_by_attrs=False*, *strip=True*)
Create an Attributes dict with target-specific entries.

Parameters

attrs [Optional[Union[str, List[Union[str, *PositionalValue*]]]]] Filter keys. If specified, only entries that match one of these keys will be returned.

target [Optional[str]] Filter targets. If specified, only entries general entries will be returned unless a target-specific entry is present, in which case it will be returned. For example,

if entries for 'class' and 'class.tex' exist and target='tex', then the 'class.tex' entry will be returned as 'class'. If target is None, then the 'class' entry will be returned.

sep [Optional[str]] The separator character (or string) to use to separate the key and target.

sort_by_attrs [Optional[bool]] If True, the returned attributes dict will have keys sorted in the same order as the attrs passed. Otherwise, the returned attributes dict will be sorted with keys in the same order as this attributes dict (default).

strip [Optional[bool]] If True, strip all target-specific terminators from the returned attris dict.

Returns

attributes [*Attributes*] A filtered attributes dict.

Examples

```
>>> attrs = Attributes('class=one tgt=default tgt.tex=tex')
>>> attrs.filter()  # python >= 3.6 should have ordered entries
Attributes{'class': 'one', 'tgt': 'default'}
>>> attrs.filter(target='tex', strip=False)
Attributes{'class': 'one', 'tgt.tex': 'tex'}
>>> attrs.filter(target='tex', strip=True)
Attributes{'class': 'one', 'tgt': 'tex'}
>>> attrs.filter(attrs='class')
Attributes{'class': 'one'}
>>> attrs.filter(attrs='tgt')
Attributes{'tgt': 'default'}
```

find_item(*attr*, *target=None*, *default=None*, *sep='.'*)

Find the key in the attributes dict with or without the target specified.

This function will not modify the keys/values in this attributes dict.

Parameters

attr [Union[str, *PositionalValue*]] The key of the attribute to retrieve or the PositionalValue to retrieve.

target [Optional[str]] If specified, search for a key for the given target. ex: class.html with be returned for the 'class' key and 'html' target, if available, otherwise, the entry for 'class' will be returned.

default [Optional[Any]] If the key is not found, return this value instead.

sep [Optional[str]] The separator character (or string) to use to separate the key and target.

Returns

key, value [Any] The retrieved item.

Examples

```
>>> attrs = Attributes('class="general" class.html="specific" pos')
>>> attrs.find_item('class')
('class', 'general')
>>> attrs.find_item('class.html')
('class.html', 'specific')
>>> attrs.find_item('class', target='html')
('class.html', 'specific')
>>> attrs.find_item('class', target='tex')
('class', 'general')
>>> attrs.find_item('pos')
('pos', <class ...StringPositionalValue'>)
>>> attrs.find_item('missing')
```

get(*attr*, *target=None*, *default=None*, *sep='.'*)

Retrieve an entry by target-specific key, if available and specified, or by key.

As opposed to the find_item method, this function will strip the target-specific terminators of returned values

Parameters

attr [Union[str, *PositionalValue*]] The key of the attribute to retrieve or the PositionalValue to retrieve.

target [Optional[str]] If specified, search for a key for the given target. ex: class.html with be returned for the 'class' key and 'html' target, if available, otherwise, the entry for 'class' will be returned.

default [Optional[Any]] If the key is not found, return this value instead.

sep [Optional[str]] The separator character (or string) to use to separate the key and target.

Returns

value [Any] The retrieved value. If the entry is a positional attribute, PositionalValue will be returned.

Examples

```
>>> attrs = Attributes('class="general" class.html="specific" pos')
>>> attrs.get('class')
'general'
>>> attrs.get('class.html')
'specific'
>>> attrs.get('class', target='html')
'specific'
>>> attrs.get('class', target='tex')
'general'
>>> attrs.get('pos')
<class ...StringPositionalValue'>
>>> attrs.get('missing')
```

property html

Format the attributes for html.

Notes

- This function won't strip target tags ('.html'), and a filter for the '.html' target should be applied before using this property

load(*s*, *sep='.'*)

Parses an attribute string into key/values for the Attributes dict.

Parameters

s: str Input string of attributes. Attributes come in two forms: positional attributes and keyword attributes (kwargs), and attributes strings have the following form: "key1=value1 key2=value2 value3"

sep [Optional[str]] The separator character (or string) to use to separate the key and target.

Examples

```
>>> attrs = Attributes()
>>> attrs.load("key1=val1 val2")
>>> attrs
Attributes{'key1': 'val1', 'val2': <class '...StringPositionalValue'>}
```

strip(*sep='.'*)

Replace the entries in this attributes dict without the target-specific terminators.

property tex_arguments

Format arguments for tex.

Notes

- This function won't strip target tags ('.tex'), and a filter for the '.tex' target should be applied before using this property

Examples

```
>>> Attributes('width=302 3').tex_arguments
'{302}{3}'
```

property tex_optionals

Format optional arguments for tex.

Notes

- This function won't strip target tags ('.tex'), and a filter for the '.tex' target should be applied before using this property

Examples

```
>>> Attributes('width=302 3').tex_optionals
'[width=302, 3]'
```

totuple()
Return a tuple for an attributes dict

Examples

```
>>> attrs = Attributes("key1=val1 val2")
>>> attrs.totuple()
(('key1', 'val1'), 'val2')
```

Classes and methods to manage tag attributes

disseminate.attributes.attributes.**is_target_specific**(*attr*, *sep='.'*)
Tests whether the given attribute is a target-specific attribute.

Parameters

attr [Union[str, *PositionalValue*]] The key of the attribute to retrieve or the PositionalValue to retrieve.

sep [Optional[str]] The separator character (or string) to use to separate the key and target.

Returns

is_target_specific [bool] True if the attr is a target-specific attribute

Examples

```
>>> is_target_specific(3)
False
>>> is_target_specific('pos')
False
>>> is_target_specific('pos.tex')
True
>>> is_target_specific('/usr/docs/text.tex')  # not quoted
True
>>> is_target_specific('"/usr/docs/text.tex"')  # quoted
False
```

disseminate.attributes.attributes.**strip_attr**(*attr*, *sep='.'*)
Strip a target-specific terminator from the given attr.

Parameters

attr [Union[str, *PositionalValue*]] The key of the attribute to retrieve or the PositionalValue to retrieve.

sep [Optional[str]] The separator character (or string) to use to separate the key and target.

Returns

stripped_attr [str] The attribute with the target-specific terminator stripped.

Examples

```
>>> strip_attr(3)
3
>>> strip_attr('143')
'143'
>>> strip_attr('width.tex')
'width'
>>> strip_attr('width')
'width'
>>> strip_attr('my_file.pdf#link-anchor')
'my_file.pdf#link-anchor'
>>> strip_attr('my_file.pdf#link-anchor.tex')
'my_file.pdf#link-anchor'
```

13.2 Builders

13.2.1 Environment

A build environment to manage and execute builders.

class disseminate.builders.environment.**Environment**(*src_filepath*, *target_root=None*, *parent_context=None*)

A project environment for rendering documents using builders.

The build project environment has the following tasks:

1. Setup the project_root and target_root
2. Setup the default decider for builders
3. Setup the default scanner for builders
4. Setup the root document.

Parameters

src_filepath [Union[str, `pathlib.Path`]] The path for the disseminate source file of the root document.

target_root [Optional[Union[str, `pathlib.Path`]]] The (optional) path for the output root directory.

parent_context [Optional[*`context.BaseContext`*]] The document context to use as the parent_context for the root document. Typically, the default_context from the settings is used as the parent context.

build(*complete=True*)

Run the build.

Parameters

complete [Optional[bool]] If True, run the build until it has completed If False, start the build in the background.

Returns

status [str] The current status of the build.

property cache_path
The path to the directory for storing cached files.

collect_target_builders(*document=None*)
Return all target builders for the root_document and all sub-documents.

Parameters

document [Optional[*document.Document*]] The document to create a root builder for.

Returns

target_builders [List[builders.Builder] The list of target builders

static create_environments(*root_path=''*, *target_root=None*, *document_extension='.dm'*)
Create environments from root documents found in the given root_path.

Parameters

root_path [Optional[Union[str, pathlib.Path]]] The path to search for root documents. By default, it is the current directory.

target_root [Optional[Union[str, pathlib.Path]]] The (optional) path for the output root directory.

document_extension [Optional[str]] The file extension for disseminate documents. ex: '.dm'

Returns

environments [List[builders.Environment]]

create_root_builder(*document=None*)
Create a root parallel builder for the target builders of the given document or the root document.

Parameters

document [Optional[*document.Document*]] The document to create a root builder for.

Returns

root_builder [composite_builders.ParallelBuilder] The root (parallel) builder.

static get_target_root(*project_root*)
Determine the target_root directory from the project_root.

property media_path
The path for to prepend to subpaths for media files.

property name
The name of the environment

13.2.2 Builder

Objects to manage builds

class disseminate.builders.builder.**Builder**(*env*, *target=None*, *parameters=None*, *outfilepath=None*, *use_cache=None*, *use_media=None*, ***kwargs*)
A build for an output file.

Parameters

env: :obj:`.builders.Environment` The build environment

parameters, args [Tuple[pathlib.Path, str, tuple, list]] The input parameters (dependencies), including filepaths, for the build

outfilepath [Optional[pathlib.Path]] If specified, the path for the output file. If not specified, an outfilepath will be automatically generated.

use_cache [Optional[bool]] If True, set the builder outfilepath into the cache_path from the builder environment. Note that this will also place temporary files created by the builder in the same directory.

use_media [Optional[bool]] If True, set the builder outfilepath subpath in the media_path in the build environment context, if specified.

Attributes

action [str] The command to execute during the build.

available [bool] Whether this builder is available to factory methods (find_builder_cls)

active_requirements [Union[tuple, bool]] If False, the builder will be inactive If a tuple of strings is specified, these conditions will be tested to see if the builder is active:

- 'priority': test that the priority attribute is an int
- **'required_execs': tests that the required_execs attribute is** specified
- 'all_execs': tests that the required execs are available

decision [builders.deciders.decider.decision] The decision object for the build, instantiate from environment's decider, to evaluate whether a build is needed.

scan_parameters_on_init [bool] If True (default), scan the parameters for additional dependencies during the __init__.

priority [int] If multiple viable builders are available, use the one with the highest priority.

required_execs [Tuple[str]] A list of external executables that are needed by the builder.

infilepath_ext [str] The format extension for the input parameters (ex: '.pdf', '.render')

outfilepath_ext [str] The format extension for the output file (ex: '.svg')

outfilepath_append [str] For automatically generated outfilepaths, the following string will be appended to the name of the file. ex: '_scale'

target [Optional[str]] If specified, use the given document target for the build. This is used in formatting the TargetPath. ex: 'html' target will store built files in the 'html/' subdirectory.

parameters_from_signals [Optional[List[str]]] A list of optional signal names to receive extra parameter dependencies.

future [Union[concurrent.futures.Future, None, str]] The future object for the process for the externally run program. The future can also be None, if a process hasn't been run.

`classmethod active()`
True if a builder is active

`build(`*complete=False*`)`
Run the build.

Parameters

complete [Optional[bool]] If True, run the build until it has completed If False, start the build in the background.

Returns

status [str] The current status of the build.

Note:

- This function will run the sub-builders.
- Each builder is atomic
- When running a build, not all of the builders might be called in the first build–for example, subsequent builders may rely on the results of previous builds. For this reason, builders should be used in conjunction with an environment to make sure a set of builds are completed or the build complete=True should be used.

`build_needed`(*reset=False*)
Decide whether a build is needed

`classmethod find_builder_cls`(*in_ext*, *out_ext=None*, *target=None*, *raise_error=True*)
Factory method to return a builder class

Parameters

in_ext: str The extension for the input file.

out_ext [str] The extension for the output file.

target [Optional[str]] The document target. ex: '.html' or '.pdf'

raise_error [Optional[bool]] If True (default), raise an exception if a builder class couldn't be found.

`get_parameter`(*name*, **parameters*)
Return parameters inserted as 2-ples of name/value.

Parameters

name The name of the 2-ple parameter (the 1st 2-ple item)

***parameters** If specified, use these parameters to get the specified parameter. Otherwise, the builder's parameters will be used.

Returns

value The corresponding 2-ple value (the 2nd 2-ple item)

`get_parameters_from_signals`(*sort=True*)
Retrieve additional parameters from emitted signals specified in the parameters_from_signals list attribute.

`property infilepaths`
Retrieve the parameters that are filepaths (`pathlib.Path`).

`property missing_parameters`
Returns True if there are no parameters or there are missing parameters.

This function will check the parameters until the parameters are all found, in which case, it will just return False.

`property not_infilepaths`
Retrieve the parameters that are not filepaths (`pathlib.Path`)

`property outfilepath`
The output filename and path

`property parameters`
The list of input parameters, including filepaths, needed for the build

run_cmd(**args*)
If the action is a external command, run it.

run_cmd_args()
Format the action, if it's a string.

Returns

run_cmd_args [Tuple[str]] A tuple of the arguments to run in a process.

static runtime_error(*future*, *error_msg=None*, *raise_error=True*)
Raise an error from a future working with subprocess

static runtime_success(*future*)
Test whether a future from a subprocess is successful.

scan_parameters()
Use the environment scanners to find additional dependencies in files specified by filepaths in the parameters.

property status
The status of the builder.

The builder can have the following states:

- 'ready': The builder is active and the parameters have been set
- 'inactive': The builder isn't active–see the active property
- 'missing (parameters)': All the required parameters have not been specified or files for paths in the parameters do not exist
- 'missing (outfilepath)': The outfilepath was not created
- 'cancelled' : The build was cancelled.
- 'building': The builder is building
- 'done': The builder is done building

class disseminate.builders.builder.**CustomFormatter**
A custom formatter class for preparing actions into command-line arguments.

static clean_field(*f*)
Clean fields used for command-line processes

13.2.3 Copy

A Builder to copy or link files.

class disseminate.builders.copy.**Copy**(*env*, *parameters=None*, *outfilepath=None*, ***kwargs*)
A builder to copy or build a file.

build(*complete=False*)
Run the build.

Parameters

complete [Optional[bool]] If True, run the build until it has completed If False, start the build in the background.

Returns

status [str] The current status of the build.

Note:

- This function will run the sub-builders.
- Each builder is atomic
- When running a build, not all of the builders might be called in the first build–for example, subsequent builders may rely on the results of previous builds. For this reason, builders should be used in conjunction with an environment to make sure a set of builds are completed or the build complete=True should be used.

`property status`
The status of the builder.

The builder can have the following states:

- 'ready': The builder is active and the parameters have been set
- 'inactive': The builder isn't active–see the active property
- 'missing (parameters)': All the required parameters have not been specified or files for paths in the parameters do not exist
- 'missing (outfilepath)': The outfilepath was not created
- 'cancelled' : The build was cancelled.
- 'building': The builder is building
- 'done': The builder is done building

13.2.4 ImageMagick

Builders that use ImageMagick's convert.

`class disseminate.builders.imagemagick.ImageMagick(`*env*, *target=None*, *parameters=None*, *outfilepath=None*, *use_cache=None*, *use_media=None*, ***kwargs*`)`

An abstract base class for ImageMagick's convert.

`class disseminate.builders.imagemagick.Tif2png(`*env*, *target=None*, *parameters=None*, *outfilepath=None*, *use_cache=None*, *use_media=None*, ***kwargs*`)`

Converter for a tif file to a png file.

`class disseminate.builders.imagemagick.Tiff2png(`*env*, *target=None*, *parameters=None*, *outfilepath=None*, *use_cache=None*, *use_media=None*, ***kwargs*`)`

Converter for a tiff file to a png file.

13.2.5 JinjaRender

A builder that renders a string to a file.

class disseminate.builders.jinja_render.**JinjaRender**(*env*, *context*, *render_ext=None*, ***kwargs*)
A builder that renders a file using Jinja2.

Note: The JinjaRender should not render the template until the build step to make sure the document and context are properly loaded first.

Parameters

env: :obj:`.builders.Environment` The build environment

parameters, args [Tuple[*paths.SourcePath*, str, tuple, list]] The input parameters (dependencies), including filepaths, for the build

outfilepath [Optional[*paths.TargetPath*]] If specified, the path for the output file.

context [dict] A context to use with the renderer.

render_ext [str] The extension for the rendered file. Either this or the outfilepath must be specified. ex: '.tex'

build(*complete=False*)
Run the build.

Parameters

complete [Optional[bool]] If True, run the build until it has completed If False, start the build in the background.

Returns

status [str] The current status of the build.

Note:

- This function will run the sub-builders.
- Each builder is atomic
- When running a build, not all of the builders might be called in the first build–for example, subsequent builders may rely on the results of previous builds. For this reason, builders should be used in conjunction with an environment to make sure a set of builds are completed or the build complete=True should be used.

---

jinja_environment()
The jinja environment.

property outfilepath
The output filename and path

property parameters
The list of input parameters, including filepaths, needed for the build

static runtime_error(*future*, *error_msg=None*, *raise_error=True*)
Raise an error from a future working with subprocess

static runtime_success(*future*)
Test whether a future from a subprocess is successful.

template()
Retrieve the template from the context

disseminate.builders.jinja_render.**context_filepaths**(*template_filepaths*)
Return a list of context modifier files (ex: context.txt) from template filepaths.

Parameters

template_filepaths [List[pathlib.Path]] A list of filepaths for templates.

Returns

filepaths [List[pathlib.Path]] A list of paths.

disseminate.builders.jinja_render.**rewrite_path**(*context*, *stub*)
A Jinja2 filter for rewriting paths, like css paths.

Note: This function will only modify the path if an existing file can be found in the target directory. If not, it will log an error and return the path unchanged. This means that the dependent .css (and other) files should have been copied to the target directory before rendering the template.

Note: This function is not sped up from using a ThreadPoolExecutor, and implementing with a ProcessPoolExecutor gives problems with pickling objects.

---

disseminate.builders.jinja_render.**template_filepaths**(*template*, *environment*)
Return a list of filepaths from a Jinja2 template object.

Parameters

template [jinja2.Template] The jinja2 template object to load filepaths for

environment [jinja2.Environment] The jinja2 environment object

Returns

filepaths [List[pathlib.Path]] A list of paths.

13.2.6 Latexmk

Builder for tex to pdf using latexmk

class disseminate.builders.latexmk.**Latexmk**(*env*, *target=None*, *parameters=None*, *outfilepath=None*, *use_cache=None*, *use_media=None*, ***kwargs*)
Compile a latex document into pdf using latexmk

Note: This builder has been disabled by default because it fails about 20% of the time with this system's ThreadPoolExecutor implementation.

13.2.7 Pdf2svg

A builder to convert from PDF to SVG

class disseminate.builders.pdf2svg.**Pdf2SvgCropScale**(*env*, *parameters=None*, *subbuilders=None*, *use_cache=None*, ***kwargs*)

A SequentialBuilder for Pdf2Svg that includes PdfCrop and ScaleSvg builders.

class disseminate.builders.pdf2svg.**Pdf2svg**(*env*, ***kwargs*)

A builder to convert from pdf to svg.

run_cmd_args()

Format the action, if it's a string.

Returns

run_cmd_args [Tuple[str]] A tuple of the arguments to run in a process.

13.2.8 Pdfcrop

A builder to crop pdf files

class disseminate.builders.pdfcrop.**PdfCrop**(*env*, ***kwargs*)

A builder to crop a pdf and form a cropped pdf.

Parameters

parameters [Tuple[pathlib.Path, str, tuple, list]] The input parameters (dependencies), including filepaths, for the build This tuple may have the 'crop' tuple value or 'crop_percentage' value specified. If specified the borders by the given single percentage or the 4 percentages for the left, bottom, right, and top margins. ex: ('crop', 10) or ('crop', (10, 20, 10,20) or ('crop_percentage', 10)

run_cmd_args()

Format the action, if it's a string.

Returns

run_cmd_args [Tuple[str]] A tuple of the arguments to run in a process.

13.2.9 Pdflatex

A builder to convert from TEX to PDF

class disseminate.builders.pdflatex.**Pdflatex**(*env*, *target=None*, *parameters=None*, *outfilepath=None*, *use_cache=None*, *use_media=None*, ***kwargs*)

Compile a latex document into pdf using pdflatex

13.2.10 PdfRender

A builder to render a tex file to pdf.

class disseminate.builders.pdfrender.**PdfRender**(*env*, *context=None*, *template=None*, *parameters=None*, *outfilepath=None*, *subbuilders=None*, *use_cache=None*, ***kwargs*)

Render a tex file and render the pdf.

13.2.11 SaveTemp

Builder to save temporary files

class disseminate.builders.save_temp.**SaveTempFile**(*env*, *context*, *save_ext=None*, ***kwargs*)

A Builder to save a string to a temporary file

Parameters

env: :obj:`.builders.Environment` The build environment

parameters, args [Tuple[pathlib.Path, str, tuple, list]] The input parameters (dependencies), including the string to save to the temporary outfilepath, for the build

outfilepath [Optional[*paths.TargetPath*]] If specified, the path for the output file.

save_ext [str] The extension for the saved temp file.

build(*complete=False*)

Run the build.

Parameters

complete [Optional[bool]] If True, run the build until it has completed If False, start the build in the background.

Returns

status [str] The current status of the build.

Note:

- This function will run the sub-builders.
- Each builder is atomic
- When running a build, not all of the builders might be called in the first build–for example, subsequent builders may rely on the results of previous builds. For this reason, builders should be used in conjunction with an environment to make sure a set of builds are completed or the build complete=True should be used.

13.2.12 ScaleSvg

A builder to scale svg

class disseminate.builders.scalesvg.**ScaleSvg**(*env*, ***kwargs*)
A builder to scale svgs.

Parameters

parameters, args [Tuple[pathlib.Path, str, tuple, list]] The input parameters (dependencies), including filepaths, for the build This tuple should have the 'scale' tuple value specified, which is the scale to increase or decrease the image. ex: ('scale', 2.0) will double the image size.

env: :obj:`.builders.Environment` The build environment

run_cmd_args()
Format the action, if it's a string.

Returns

run_cmd_args [Tuple[str]] A tuple of the arguments to run in a process.

13.2.13 SvgRender

A builder to render a tex file and convert it to an svg.

class disseminate.builders.svgrender.**SvgRender**(*env*, *parameters=None*, *outfilepath=None*, *context=None*, *template=None*, *subbuilders=None*, *use_cache=None*, ***kwargs*)
Render a tex file and render the svg.

13.2.14 XHtml2Epub

Builders for EPUB files

class disseminate.builders.xhtml2epub.**XHtml2Epub**(*env*, *context*, ***kwargs*)
Convert xhtml files to an epub v3 file.

Parameters

parameters, args [Tuple[pathlib.Path, str, tuple, list]] The input parameters (dependencies), including filepaths for xhtml files, for the build. The parameters must include a 'toc.xhtml' file for the Table of Contents.

build(*complete=False*)
Run the build.

Parameters

complete [Optional[bool]] If True, run the build until it has completed If False, start the build in the background.

Returns

status [str] The current status of the build.

Note:

- This function will run the sub-builders.

- Each builder is atomic
- When running a build, not all of the builders might be called in the first build–for example, subsequent builders may rely on the results of previous builds. For this reason, builders should be used in conjunction with an environment to make sure a set of builds are completed or the build complete=True should be used.

---

create_opf_builder(*template_name='default/xhtml/content.opf'*)
Create a render builder for the content.opf file.

Parameters

template_name [Optional[str]] The name of the template to use for the content.opf.

filepath_dict_list(*filepaths*, *suffix*)
Create a list of dicts from the given filepaths

property parameters
The list of input parameters, including filepaths, needed for the build

13.2.15 Executor

The pool executor for running multiple functions at once.

disseminate.builders.executor.**run**(*timeout*, ***kwargs*)
Run the command with the given arguments.

13.2.16 Exceptions

Build exceptions and utilities for build exceptions

exception disseminate.builders.exceptions.**BuildError**
A error was encountered in conducting a build.

13.2.17 utils

Utilities for builders and environments

disseminate.builders.utils.**generate_mock_parameters**(*env*, *parameters*, *project_root=None*, *subpath=None*, *ext=None*, *context=None*, *gen_hash=True*)
Generate a mock set of parameters.

This function is used by builders, like JinjaRender and SaveTempFile, that do not use an input file but need to save to an output file. It includes an option to rename the filename with a hash.

Parameters

env [builders.Environment] The build environment.

parameters [Tuple[*paths.SourcePath*, str, tuple, list]] The parameters for the builder.

project_root [Optional[Union[paths.SourcePath, str]]] If given, use the specified project root for the mock infilepath.

subpath [Optional[Union[paths.SourcePath, str]]] If given, use the specified subpath for the mock infilepath.

ext [Optional[str]] If given, use the specified extension for the mock infilepath.

gen_hash [Optional[bool]] If True, append the hash from the given parameters to append to the infilepath filename.

disseminate.builders.utils.**generate_outfilepath**(*env*, *parameters*, *target=None*, *append=None*, *ext=None*, *use_cache=False*, *use_media=False*)

Given a set of parameters, generate an outfilepath.

Parameters

env [`builders.Environment`] The build environment.

parameters [Tuple[`pathlib.Path`]] The parameters for the builder. The first pathlib.Path will be used.

target [Optional[str]] If specified, use the given target as a subdirectory in the target_root.

append [Optional[str]] If specified, append the given string to the returned filename

ext [Optional[str]] If specified, return a cache_filepath with the given target format.

use_cache [bool] If True, return a path in the cache directory.

use_media [bool] If True, return a path with a subpath prepended with the media_path. ex: 'media/'

Returns

outfilepath [Union[*paths.TargetPath*, None]] The outfilepath based on the parameters given, or None if no outfilepath could be generated

---

Note: When dealing with absolute paths, only the filename is kept in formulating the subpath. So for

target_root = '.' target = 'html' use_media = True parameters = ['/usr/local/bin/2to3']

The generated outfilepath will be:

TargetPath('html/media/2to3')

---

disseminate.builders.utils.**sort_key**(*parameter*)

Sort key function for a series of parameters to give a consisting ordering.

13.2.18 CompositeBuilder

class disseminate.builders.composite_builders.composite_builder.**CompositeBuilder**(*env*, *subbuilders=None*, ***kwargs*)

A builder that integrates multiple (sub)-builders

Parameters

subbuilders [Optional[List[`builders.Builder`]] The subbuilders to run as part of this composite builder.

Attributes

clear_done [bool] If True (default), remove 'done' subbuilders during the build.

build(*complete=False*)

Run the build.

Parameters

complete [Optional[bool]] If True, run the build until it has completed If False, start the build in the background.

Returns

status [str] The current status of the build.

Note:

- This function will run the sub-builders.
- Each builder is atomic
- When running a build, not all of the builders might be called in the first build–for example, subsequent builders may rely on the results of previous builds. For this reason, builders should be used in conjunction with an environment to make sure a set of builds are completed or the build complete=True should be used.

`flatten`(*builder=None*)
Generate a flat list with this builder and all subbuilders (recursively).

Parameters

builder [Optional[:obj:` <.composite_builders.CompositeBuilder>`]] If specified, flatten the given builder, instead of this builder.

Returns

flattened_list [List[:obj:` <.composite_builders.CompositeBuilder>`]] The flattened list of builders.

`futures`(*builder=None*)
Generate a list of this builder's future and all sub-builder futures (recursively)

Parameters

builder [Optional[:obj:` <.composite_builders.CompositeBuilder>`]] If specified, flatten the given builder, instead of this builder.

Returns

futures [List[:obj:` <concurrent.futures.Future>`]] The flattened list of futures.

`print`(*level=1*, *max_level=None*)
Print the builder and subbuilders

`run_cmd_args`()
Format the for all sub commands

Returns

run_cmd_args [Tuple[str]] A tuple of the arguments for all sub-builders

`property status`
The status of the builder.

The builder can have the following states:

- 'ready': The builder is active and the parameters have been set
- 'inactive': The builder isn't active–see the active property

- 'missing (parameters)': All the required parameters have not been specified or files for paths in the parameters do not exist
- 'missing (outfilepath)': The outfilepath was not created
- 'cancelled' : The build was cancelled.
- 'building': The builder is building
- 'done': The builder is done building

13.2.19 SequentialBuilder

class disseminate.builders.composite_builders.sequential_builder.**SequentialBuilder**(*env*, ***kwargs*)

A composite builder that runs subbuilders in sequence (i.e. wait for one to finish before starting the next)

Notes

With chain_on_create is enabled, he build filepaths for subbuilders are set as follows, with user-supplied paths in parentheses:

```
builder => subbuilder1 (parameters) => outfilepath1
        => subbuilder2 outfilepath2 => outfilepath3
        => subbuilder3 outfilepath3 => outfilepath4
        => outfilepath4 => (outfilepath)
```

Attributes

chain_on_create [bool] If True (default), chain the parameters and outfilepath of the subbuilders to follow each other.

copy [bool] If True (default), the last subbuilders will be a Copy build to copy the result to the final outfilepath. This only applies if use_cache is False, in which case the final generated file is copied from the cache directory to the target directory.

subbuilder_for_outfilename [Optional[builders.Builder]] The subbuilder instance to use for generating the outfilepath of the SequentialBuilder. Specifying this class name is useful for SequentialBuilders that use the filename hash generated by some subbuilders, like JinjaRender or SaveTempFile.

chain_subbuilders()

Chain the parameters and outfilepath of the subbuilders to follow each other

property outfilepath

The output filename and path

13.2.20 ParallelBuilder

class disseminate.builders.composite_builders.parallel_builder.**ParallelBuilder**(*env*, *subbuilders=None*, ***kwargs*)

A composite builder that runs subbuilders in parallell (i.e. run the subbuilders together at the same time)

add_build(*parameters*, *outfilepath=None*, *target=None*, *context=None*, *in_ext=None*, *out_ext=None*, *builder_cls=None*, ***kwargs*)

Add a subbuilder to the ParallelBuilder.

Parameters

parameters, args [Tuple[`pathlib.Path`, str, tuple, list]] The input parameters (dependencies), including filepaths, for the build

outfilepath [Optional[str, `pathlib.Path`]] The path for the output file.

target [Optional[str]] The document target for the build. ex: 'html' or 'tex'

context [Optional[*context.BaseContext*]] The document context dict that owns the parallel builder.

in_ext [Optional[str]] The extension for the input file. This is used to find the correct builder with the find_builder_cls method. If this is not specified, an filepath in the parameters should be specified.

out_ext [Optional[str]] The extension for the input file. This is used to find the correct builder with the find_builder_cls method. If this is not specified, the outfilepath should be specified.

builder_cls [Optional[`builders.Builder`]] The builder class to use in create a new build.

****kwargs** Optional keyword arguments to use in creating the builder.

Returns

builder [`builders.Builder`] The newly created builder.

build_needed(*reset=False*)

Decide whether a build is needed

13.2.21 TargetBuilder

A builder for document targets.

class disseminate.builders.target_builders.target_builder.**TargetBuilder**(*env*, *context*, *parameters=None*, *outfilepath=None*, *subbuilders=None*, *use_cache=None*, ***kwargs*)

A builder for a document target, like html, tex or pdf

Parameters

env: :obj:`.builders.Environment` The build environment

context: :obj:`.context.Context` The context dict for the document being rendered.

parameters, args [Tuple[`pathlib.Path`, str, tuple, list]] The input parameters (dependencies), including filepaths, for the build

outfilepath [Optional[pathlib.Path]] If specified, the path for the output file.

subbuilders [List[builders.Builder]] A list of subbuilders to add to this TargetBuilder.

Attributes

only_root [Optional[bool]] If True, only add the target builder for the root document.

add_parallel_builder [Optional[bool]] If True (default), create a parallel builder for adding dependencies for a target.

add_render_builder [Optional[bool]] If True (default), create a render builder to render the target.

`add_build`(*parameters*, *outfilepath=None*, *context=None*, *use_cache=None*, ***kwargs*)
Create and add a sub-builder to the composite builder.

`build`(*complete=False*)
Run the build.

Parameters

complete [Optional[bool]] If True, run the build until it has completed If False, start the build in the background.

Returns

status [str] The current status of the build.

Note:

- This function will run the sub-builders.
- Each builder is atomic
- When running a build, not all of the builders might be called in the first build–for example, subsequent builders may rely on the results of previous builds. For this reason, builders should be used in conjunction with an environment to make sure a set of builds are completed or the build complete=True should be used.

`chain_subbuilders`()
Chain the parameters and outfilepath of the subbuilders to follow each other

`property outfilepath`
The output filename and path

13.2.22 EpubBuilder

A CompositeBuilder for html files.

`class` disseminate.builders.target_builders.epub_builder.`EpubBuilder`(**args*, ***kwargs*)
A builder for epub files.

`create_subbuilders`()
Create subbuilders needed by this builder

`create_toc_xhtml_builder`(*context=None*, *template_name='default/xhtml/toc.xhtml'*)
Create a toc.xhtml file for the document.

Parameters

context [document.DocumentContext] The document context for the document that owns this builder.

outfilepath [Optional[paths.TargetPath]] The outfilepath to use for the toc.xhtml

template_name [Optional[str]] The name of the template file to use in making the toc.

find_or_create_xhtml_builders(***kwargs*)
Find or create the target XHtmlBuilder for all documents in the document tree.

has_toc_xhtml(*xhtml_filepaths=None*)
Evaluates whether there is a toc.xhtml file in the xhtml files.

xhtml_filepaths(*xhtml_builders=None*)
Return a flat list of xhtml, css, svg filepaths from the XHtmlBuilders

13.2.23 HtmlBuilder

class disseminate.builders.target_builders.**HtmlBuilder**(*env*, *context*, ***kwargs*)
A builder for html files.

13.2.24 PdfBuilder

A TargetBuilder for pdf files.

class disseminate.builders.target_builders.pdf_builder.**PdfBuilder**(*env*, *context*, *parameters=None*, *outfilepath=None*, *subbuilders=None*, ***kwargs*)
A builder for Pdf files.

add_build(*parameters*, *outfilepath=None*, *context=None*, ***kwargs*)
Create and add a sub-builder to the composite builder.

13.2.25 TexBuilder

A TargetBuilder for tex files.

class disseminate.builders.target_builders.tex_builder.**TexBuilder**(*env*, *context*, *parameters=None*, *outfilepath=None*, *subbuilders=None*, *use_cache=None*, ***kwargs*)
A builder for tex files.

13.2.26 TxtBuilder

A TargetBuilder for txt files.

class disseminate.builders.target_builders.txt_builder.**TxtBuilder**(*env*, *context*, *parameters=None*, *outfilepath=None*, *subbuilders=None*, *use_cache=None*, ***kwargs*)

A builder for txt files.

13.2.27 XHtmlBuilder

A TargetBuilder for xhtml files.

class disseminate.builders.target_builders.xhtml_builder.**XHtmlBuilder**(*env*, *context*, ***kwargs*)

A builder for xhtml files.

13.2.28 Receivers

Receivers for TargetBuilders

disseminate.builders.target_builders.receivers.**add_file**(*parameters*, *context*, *in_ext*, *target*, *use_cache=False*)

Add a file to the target builder.

Parameters

parameters [Union[str, List[str, pathlib.Path]]] The parameters for the build

context [*BaseContext*] The document context dict

in_ext [str] The extension for the file.

target [str] The document target to add the file for.

use_cache [Optional[bool]] If True, use cached paths when adding files.

Returns

outfilepath [pathlib.Path] The outfilepath for the file from the build.

disseminate.builders.target_builders.receivers.**add_target_builders**(*root_document*)

Add target builders to a document context

disseminate.builders.target_builders.receivers.**build**(*document*, *complete=True*)

Build a document tree's targets (and subdocuments) using the target builders.

disseminate.builders.target_builders.receivers.**build_needed**(*document*)

Evaluate whether any of the target builders need to be build

disseminate.builders.target_builders.receivers.**find_builder**(*context*, *target=None*, ***kwargs*)

Find a target builder in a document context, or None if None was found.

Parameters

context [document.DocumentContext] The context dict for the document.

target [Optional[str]] The target to retrieve the target builder for.

Returns

builders [List[builders.target_builders.TargetBuilder]] Returns a list of all matching target builders.

13.2.29 Signals

Signals for target builders

disseminate.builders.target_builders.signals.**add_file** = <**disseminate.signals.signals.Signal object**>
Add a file dependency to a target builder. Takes parameters, context, in_ext, target and use_cache.

disseminate.builders.target_builders.signals.**find_builder** = <**disseminate.signals.signals.Signal object**>
Given a document context and a target, find the corresponding target builder.

13.2.30 Exceptions

Exceptions for target builders

exception disseminate.builders.target_builders.exceptions.**TargetBuilderNotFound**
Error raised when a target builder was requested but not found.

13.2.31 Decider

Decider classes to evaluate whether a build is needed.

class disseminate.builders.deciders.decider.**Decider**(*env*)
A decider to evaluate whether a build is needed.

property decision
Return the Decision class associated with this decider.

decision_cls
alias of *disseminate.builders.deciders.decider.Decision*

class disseminate.builders.deciders.decider.**Decision**(*parent_decider*)
A decision to build.

The base decision checks to see if the input files and output file exist.

Parameters

parent_decider [builders.decider.Decider] The parent decider instance that created this decision.

build_needed(*inputs*, *output*, *reset=False*)
Determine whether a build is needed.

Parameters

inputs [List[str, *paths.SourcePath*, tuple]] The input infilepaths, strings and arguments to use in the build.

output [*paths.TargetPath*] The outfilepath for the built file

reset [Optional[bool]] If True, reset cached values in determining whether the build is needed.

Raises

MissingInputFiles Raise if one or more of the input files are missing.

13.2.32 Md5Decider

A decider that uses MD5 hashes.

class disseminate.builders.deciders.md5decider.**Md5Decider**(*env*, *name=None*)
A decider that uses md5 hashes to compute whether a build is needed.

property **db_path**
The path for the database file

decision_cls
alias of *disseminate.builders.deciders.md5decider.Md5Decision*

class disseminate.builders.deciders.md5decider.**Md5Decision**(*parent_decider*)
A decision for the Md5Decider

build_needed(*inputs*, *output*, *reset=False*)
Determine whether a build is needed.

Parameters

inputs [List[str, *paths.SourcePath*, tuple]] The input infilepaths, strings and arguments to use in the build.

output [*paths.TargetPath*] The outfilepath for the built file

reset [Optional[bool]] If True, reset cached values in determining whether the build is needed.

Raises

MissingInputFiles Raise if one or more of the input files are missing.

static **calculate_hash**(*inputs*, *output*)
Calculate the md5 hash for the inputs, output and args.

13.2.33 util_hash

Utilities for hashing string and files.

disseminate.builders.deciders.utils_hash.**hash_file**(*filepath*, *chunk_size*, *hashfunc*)
Create a unique hash for the file contents of the given filepath.

Parameters

filepath [pathlib.Path] The filepath of the file whose contents will be hashed.

chunk_size [Optional[int]] When reading the contents of files (from pathlib.Path items), read the files in the given number of chunk bytes.

hashfunc [Optional[func]] The type of hash to use.

Returns

hash [bytes] The hash bytes.

disseminate.builders.deciders.utils_hash.**hash_items**(**items*, *chunk_size=4096*, *hashfunc=<built-in function openssl_md5>*, *sort=True*)
Create a unique text string hash from the given item objects.

Parameters

***items** [Tuple[obj, str, bytes, `pathlib.Path`]] Items to use in calculating the hash.

chunk_size [Optional[int]] When reading the contents of files (from `pathlib.Path` items), read the files in the given number of chunk bytes.

hashfunc [Optional[func]] The type of hash to use.

sort [Optional[bool]] If True, sort the items before calculating the hash. Enabling this option ensures that the items order does not change the hash.

Returns

hashdigest [str] The hash digest string.

`disseminate.builders.deciders.utils_hash.`**hash_pdf**(*filepath*, *chunk_size*, *hashfunc*, *startswith=(b'/CreationDate', b'/ModDate', b'/ID')*)

Create a unique hash for the file contents of the given pdf filepath.

PDF files are parsed differently because they contain metadata on the date created, which may not reflect a change in the actual content of the file.

Parameters

filepath [`pathlib.Path`] The filepath of the file whose contents will be hashed.

chunk_size [Optional[int]] When reading the contents of files (from `pathlib.Path` items), read the files in the given number of chunk bytes.

hashfunc [Optional[func]] The type of hash to use.

startswith [Optional[Tuple[str]]] Lines that start with the given bytes will be ignored in the hash.

Returns

hash [bytes] The hash bytes.

`disseminate.builders.deciders.utils_hash.`**line_splitter**(*file*, *newline*, *chunk_size*, *tail=None*)

Given a file object, read its contents in chunks into lines without breaking newlines.

13.2.34 Scanner

A scanner object to find implicit dependencies.

`class disseminate.builders.scanners.scanner.`**Scanner**

A scanner object parses the contents of a file and finds implicit file dependencies.

classmethod get_scanner(*extension*)

Get the scanner for the given extension

classmethod scan(*parameters*, *raise_error=True*)

Scan parameters for dependencies.

Parameters

str_or_filepath [Union[str, `pathlib.Path`]] A string or a filepath to a filename to scan.

paths [List[`pathlib.Path`]] A list of directories to search.

raise_error [bool] If True (default), raise a FileNotFoundError if a dependency file could not be found.

Returns

new_infilepaths [List[*paths.SourcePath*]] A list of infilepath dependencies.

static scan_function(*content*)
The function to scan content for new infilepaths.

Subclasses derive this function.

13.2.35 HtmlScanner

A scanner for html files.

class disseminate.builders.scanners.html_scanner.**HtmlScanner**
A scanner for html files.

static scan_function(*content*)
Scan additional file dependencies from the content of an html file.

13.3 Checkers

Software checkers for external dependencies.

Checkers are used to verify the installation and configuration of external software dependencies.

13.3.1 Checker

The Checker base class for checking dependencies.

class disseminate.checkers.checker.**Checker**(*category*, **dependencies*)
Bases: *disseminate.checkers.types.All*

Check for installed dependencies.

Note: The check implements checker handlers for different categories of dependencies (check_ methods).

For example, a dependencies listing with a category 'executables' will run the *check_executables()* method and a listing with a category of 'packages' will run the *check_packages()* method.

---

check_executables(*executables=None*)
Check the availability of executables.

Parameters

executables [Union[List[str], Tuple[str]]] The list of executables to check for availability.

check_packages(*packages=None*)
Check the availability of packages.

Parameters

packages [Union[List[str], Tuple[str]]] The list of packages to check for availability.

classmethod checker_subclasses()
All subclasses checkers.

13.3.2 Types

Bases classes for checkers of software dependencies.

class disseminate.checkers.types.**All**(*category*, **dependencies*)
Bases: *disseminate.checkers.types.SoftwareDependencyList*

A list for external programs and packages in why all of them should be present.

class disseminate.checkers.types.**Any**(*category*, **dependencies*)
Bases: *disseminate.checkers.types.SoftwareDependencyList*

A list for external programs and packages in why any of them may be present.

class disseminate.checkers.types.**Optional**(*category*, **dependencies*)
Bases: *disseminate.checkers.types.SoftwareDependencyList*

A list for external programs and packages in why none of them are required to be present.

class disseminate.checkers.types.**SoftwareDependency**(*name*, ***kwargs*)
Bases: object

An external software program or package dependency.

Parameters

name [str] The name of the external program or package.

Attributes

path [str] The path for the external program or package.

available [Optional[bool]] True, if dependency is available, False otherwise. May also be None if the availability is not determined yet

class disseminate.checkers.types.**SoftwareDependencyList**(*category*, **dependencies*)
Bases: object

A listing of software dependencies

Parameters

category [str] The category name for the software dependency list.

***dependencies** [Tuple[Union[str, *SoftwareDependencyList*]] The listing of dependency names or sub dependency lists.

flatten(*items=None*, *level=1*)
Return a flattened list of software dependency objects.

The flattened list includes SoftwareDependency and SoftwareDependencyList objects.

Returns

flattened_list [List[Tuple[int, Union[*SoftwareDependency*, *SoftwareDependencyList*]]]] A flattened list of tuples with the level (int) and software dependencies or software dependency lists.

keys()
Name or category of sub-dependencies.

13.3.3 PdfChecker

Checkers for .tex targets.

class disseminate.checkers.pdf.**PdfChecker**(*pdf_deps=All[All[Any[pdflatex(unknown), xelatex(unknown), lualatex(unknown)], Any[kpsewhich(unknown)]], All[graphicx(unknown), caption(unknown), amsmath(unknown), mathtools(unknown), bm(unknown), easylist(unknown), fancyvrb(unknown), hyperref(unknown), enumitem(unknown), geometry(unknown), xcolor(unknown)], Optional[ecrm1200(unknown), tcrm1200(unknown)], Optional[article(unknown), report(unknown), tufte - book(unknown)]]*)

Bases: *disseminate.checkers.checker.Checker*

Checker for pdf renderings from latex.

check_classes(*classes=None*)

Check the availability of classes.

Parameters

classes [Union[List[str], Tuple[str]]] The list of packages to check for availability.

check_classes_kpsewhich(*classes=None*)

Check available latex classes (.cls) using kpsewhich.

Parameters

classes [Union[List[str], Tuple[str]]] The list of classes to check for availability.

check_fonts(*fonts=None*)

Check available of latex fonts (.tfm) using kpsewhich.

Parameters

fonts [Union[List[str], Tuple[str]]] The list of fonts to check for availability.

check_kpsewhich(*listing*, *ext*)

Check the availability of latex packages using kpsewhich.

Parameters

listing [Union[List[str], Tuple[str]]] Base filenames for latex classes or packages to check. ex: 'article'

ext [str] The extension to search. ex: '.sty' or '.cls'

check_packages_kpsewhich(*packages=None*)

Check available latex packages (.sty) using kpsewhich.

Parameters

packages [Union[List[str], Tuple[str]]] The list of packages to check for availability.

13.3.4 Python Checker

Checkers for python packages.

class disseminate.checkers.python.**PythonChecker**(*python_deps=All[Any[python3.6(unknown), python3.7(unknown), python3.8(unknown), python3.9(unknown)], All[regex>=2018.11.22(unknown), jinja2>=2.11(unknown), lxml>=4.3.0(unknown), python-slugify>=2.0.1(unknown), pdfCropMargins>=0.1.4(unknown), click>=7.0(unknown), tornado>=6.1(unknown), pygments >=2.6(unknown), diskcache>=4.1(unknown), pathvalidate>=2.2(unknown)]]*)

Bases: *disseminate.checkers.checker.Checker*

Checker for python dependencies.

Attributes

freeze [Optional[Tuple[str]]] If pip is available, this is populated with a tuple of strings for installed packages.

check_packages_pip(*packages=None*)

Check the availability of specific python packages in pip.

Parameters

listing [Union[List[str], Tuple[str]]] Packages names with optional version specifier.

13.3.5 External Checkers

Checkers for external dependencies.

class disseminate.checkers.external.**ImageExtChecker**(*image_deps=Optional[Optional[asy(unknown), convert(unknown), pdf2svg(unknown), pdf - crop - margins(unknown), rsvg - convert(unknown)]]*)

Bases: *disseminate.checkers.checker.Checker*

Checker for external dependencies for image processing.

13.3.6 Exceptions

Exceptions for check classes.

exception disseminate.checkers.exceptions.**MissingHandler**

Bases: disseminate.checkers.exceptions.Checker

A handler for a specified category is not available to the checker.

13.3.7 utils

Utilities for checkers.

disseminate.checkers.utils.**name_and_version**(*string*)

Parse the name and version from a package name string with an optional specifier.

Parameters

string [str] The string with a name and version to parase.

Returns

name, operator, version [Union[None, Tuple[str, Union[operator, None], Union[None, Tuple[int]]]] The name, operator and version for a package. None, if the string couldn't be parsed.

Examples

```
>>> name_and_version('test')
('test', None, None)
>>> name_and_version('test-ab_one')
('test-ab_one', None, None)
>>> name_and_version('test-ab_one>=0.3.1')
('test-ab_one', <built-in function ge>, (0, 3, 1))
>>> name_and_version('new-package>1.2')
('new-package', <built-in function gt>, (1, 2))
>>> name_and_version('jinja2>=2.10')
('jinja2', <built-in function ge>, (2, 10))
```

13.4 CLI

13.4.1 Commands

dm

Disseminate Document Processor

```
dm [OPTIONS] COMMAND [ARGS]...
```

Options

--version

Show the version and exit.

--debug

Show debugging information

build

Build a disseminate project

```
dm build [OPTIONS]
```

Options

-i, --in-path <in_path>
: the directory or file path for a project root document

-o, --out-dir <out_dir>
: the target directory for the generated output documents

-p, --progress
: Show a progress bar for the build

init

Initialize a new template project

```
dm init [OPTIONS] [NAMES]...
```

Options

-o, --out-dir <out_dir>
: the directory to create the project starter

--info
: Show detailed information on the specified project starter

-l, --list
: List the available project starters

-n, --non-interactive
: Do not ask questions

Arguments

NAMES
: Optional argument(s)

preview

Preview documents with a local webserver

```
dm preview [OPTIONS]
```

Options

-i, --in-path <in_path>
the directory or file path for a project root document

-p, --port <port>
The port to listen to for the webserver

Default 8899

--debug
Show debugging information

setup

Setup and configuration options

```
dm setup [OPTIONS]
```

Options

--check
Check required executables and packages

--list-signals
List the available signals

13.4.2 Setup - Checkers

Command-line interface tools for Checkers

disseminate.cli.setup.checkers.**print_checkers**()
Print the availabilitiy of executables and packages from the checkers.

disseminate.cli.setup.checkers.**print_single_check**(*msg*, *status*, *color=None*, *bold=False*, *spacer=' '*, *level=1*)
Print a single check line.

13.4.3 Setup - Signals

Command-line interface tools for Signals

disseminate.cli.setup.signals.**print_signals**(*signal_namespace*)
Given an signal namespace, print all signals and connected receivers.

Parameters

signal_namespace [*signals.signals.Namespace*] The namespace for all signals.

Returns

None

13.4.4 Terminal Utilities

Terminal utilities in the CLI.

disseminate.cli.term.**fill_string**(*start*, *end*, *start_len=None*, *end_len=None*, *spacer=' '*, *width=None*)
Prints a start string and end string to fill a line on the terminal using the spacer character.

disseminate.cli.term.**term_width**()
Retrieve the current width of the terminal.

13.5 Contexts

Contexts are dictionaries (dicts) that contain the entries of the header for a disseminate document, the body of the disseminate document ('body' entry) and other default settings. The context is used directly in rendering documents.

The context has the following responsibilities:

1. *Variable storage*. The context is a dict that contains the values needed to properly render a target document.
2. *Data validation*. Specific entries, like the src_filepath, must be a specific type, like a `SourcePath`. When initializing and resetting a context, the context will check that entries match this type, and it will convert to the valid type if it can.
3. *Inheritance*. A document and its sub-documents will each have their own context. It is useful to have the context of sub-documents inherit from the parent document's context (parent context) for some values. A context keeps a reference to the parent context, and when resetting the context, some (or all) of the parent context values are either copied over or referenced.
4. *Intelligent resetting*. A context intelligently resets to its initial values and the values of the parent_context on reset. These values can then be overwritten when the document source is read in.

13.5.1 Base Context

The base class for Context objects that include functions to validate a context.

class disseminate.context.context.**BaseContext**(**args*, ***kwargs*)
Bases: `dict`

A context dict with entries used for rendering target documents.

Contexts are suppose to be data container dicts–i.e. there are no sophisticated processing or rendering functions. The available functions only manage the data and set of the context.

The BaseContext is basically a heritable dict. It keeps track of dict lineage and initial values so that it can be reset to its initialized state.

Note: I've tried different implementations, including a ChainMap-like inheritance. The problem with these is that key lookup and __contains__ lookup are relatively slow, compared to a standard dict. For this reason, I've decided to implement the BaseContext as a simple dict with values copied from a parent_context. The downside of this approach is that it's more memory intensive, but this is mitigated to a great extended from creating shallow copies or weak references of parent_context entries.

Parameters

parent_context [Optional[dict]] A parent or template context.

***args, **kwargs** [tuple and dict] The entries to populate in the context dict.

Examples

```
>>> parent = BaseContext(a=1, b=[])
>>> child = BaseContext(parent_context=parent)
>>> child['a']
1
>>> child['a'] = 2
>>> child['a']
2
>>> parent['a']
1
>>> child['b'].append(1)
>>> child['b']
[1]
>>> parent['b']
[]
```

Attributes

validation_types [Dict[str, type]] A listing of entry keys and the types they should be. If None is listed as the type, then a type check will not be conducted.

parent [*BaseContext*] A weakref to a BaseContext of a parent document.

do_not_inherit [Set[str]] The context keys that should not be accessed (inherited) from the parent context.

exclude_from_reset [Set[str]] The context entries that should not be removed when the context is reset.

replace [Set[str]] The *muttable* context entries that should be replaced, rather than appended, from the parent context.

_initial_values [dict] A dict containing the initial values. Since this starts with an underscore, it is hidden when listing keys with the keys() function.

_parent_context [*Type[BaseContext]*] A dict containing the parent context from which values may be inherited. Since this starts with an underscore, it is hidden when listing keys with the keys() function.

`filter`(*keys*)

Create a new context with the entries copied for the given keys.

Parameters

keys [Iterable[str]] The keys for entries to include in the returned filtered BaseContext

Returns

copy [*context.BaseContext*] The filtered copy of the BaseContext.

`classmethod find_do_not_inherit`()

Retrieve a union set for the do_not_inherit attribute of this class and all parent classes.

Returns

do_not_inherit [Set[str]] A set of all attributes that should not be inherited by child class instances.

`is_valid`(**keys*, *must_exist=True*)

Validate the entries in the context dict.

This function checks that the keys denoted by the validate_types class attribute match the corresponding types for its values.

Parameters

keys [Optional[Tuple[str]]] If specified, only the given keys will be checked if they're in the validate_types class attribute. Otherwise all validate_types will be checked.

must_exist [Optional[bool]] If True (default), then the entry must also exist in addition to having the correct type. If False, then entries can be missing in this dict that are listed in the validate_types class attribute.

Returns

validated [bool] True if the context is valid. False if the context is not valid

`load`(*string*, *strip_header=True*, *overwrite=True*)

Load context entries from a string.

The load function interprets a string containing a dict in the format processed by `str_to_dict`. Additionally, this function parses header portions of strings delineated by 3 or more '-' characters, and only new entries entries that match the types of existing entries are inserted.

Parameters

string [str] The string to load into this context.

strip_header [Optional[bool]] If True (default), the returned string will have the header removed.

overwrite [Optional[bool]] If True, allow existing entries in self to be overwritten by changes.

Returns

string [str] The processed string with the header removed (if available) or just the string passed as an argument.

Examples

```
>>> context = BaseContext()
>>> rest = context.load('test: 1')
>>> print(context)
BaseContext{test: 1}
```

`match_update`(*changes*, *keys=None*, *overwrite=True*, *level=1*)

Update this context dict with changes by matching entry types.

Matched update behaves like a dict's update method with these key differences:

1. If changes is a string, it is first converted to a dict.
2. Existing entries from changes are converted to the same type as the value in this dict. See examples below.
3. Mutable values are deep copied from the changes dict.

Parameters

changes [Union[str, dict, *BaseContext*]] The changes to update into this context dict.

keys [Optional[Iterable[str]]] If specified, only update the given keys

overwrite [Optional[bool]] If True, allow existing entries in self to be overwritten by changes.

level [Optional[int]] The level of recursion for this function.

Notes

The matched update only updates entries for matching types and appends to nested mutables, like lists and dicts.

1. New entries from 'changes' are copied
   a. If the change's value has a 'copy' method, a copy is created with that method
   b. Otherwise, the value is copied directly
2. Existing entries are converted to the type of the value in this context dict.
   a. Lists, sets and dicts: A copy of the changes value is converted to the respective type and appended to this context dict's list, set or dict.
   b. Immutables are converted and added directly.

Examples

```
>>> orig = BaseContext(test = [])
>>> orig.match_update(changes={'test': 'test'})
>>> orig['test']
['test']
>>> orig.match_update(changes={'test': 'more changes'})
>>> orig['test']
['more changes', 'test']
```

property parent_context
The parent context.

print()
Pretty print this context

reset()
(Selectively) resets the context to its initial state.

The context is reset by removing items with keys not specified in the 'exclude_from_clear' class attribute.

Note: Entries in the parent_context are copied to this context. If the parent_context entry is a mutable, like a list or dict, then a copy of that mutable is created if a 'copy' method exists for that mutable. However, mutables within those mutables will still point to the original.

Examples

```
>>> l1, l2 = ['a'], ['b']
>>> context = BaseContext(l1=l1)
>>> context['l2'] = l2
>>> context['l1'].append(1)
>>> context['l2'].append(2)
>>> context.reset()
>>> context['l1'] == ['a']  # reset to original l1
True
>>> 'l2' in context
False
```

exception disseminate.context.context.**ContextException**
Bases: Exception

An exception raised when processing a BaseContext

13.5.2 Utilities

Utilities for contexts.

disseminate.context.utils.**find_header_entries**(*context*)
Find context entries that contain a header string.

Parameters

context [dict] The context dict to search.

Returns

keys The keys for entries with header entries.

disseminate.context.utils.**load_from_string**(*string*)
Load a context from a string containing a header string.

Parameters

string [str] The string to load into this context.

Returns

string, dict [Tuple[str, dict]] The processed string with the header removed and the dict with the loaded values from the header.

13.6 Document

A document corresponds to a single disseminate source file. A document may contain sub-documents, which in turn may hold sub-documents, to form a tree. The top-level document is known as a *root document* and the root document and all sub-documents are known as a project.

13.6.1 Responsibilities

A document has the following responsibilities:

1. *Managers*. The root document is responsible to create managers. There should only be one of each type of manager in a project, and a project uses the same managers.
2. *Rendering*. The document is responsible for delegating rendering of the document in the target format(s).
3. *Processors*. When rendering a document, the 'body' entry in the context (*i.e.* the body of the disseminate source file) will be run through a list of processors, listed in the document class.
4. *Conversion*. Some documents types must be compiled or converted to their final format–*e.g.* to produce `pdf` targets, a `tex` target must be converted using `pdflatex`. The document manages this conversion as part of the rendering process.
5. *Context*. A document owns a context that is populated by the disseminate file header, and which holds the variables used in rendering a document into a target format.
6. *Paths*. The document manages paths for the disseminate source file and the target files.

13.6.2 Definitions

1. *Root document*. A document that is not included as a sub-document to another document. The root document's context will have objects, like the label manager, that are shared by its sub-documents in a project.
2. *Project*. A root document and zero or more sub-documents forming a document tree. Sub-documents are included by `include` statements in the header of the root document.
3. *doc_id*. The document identifier for a document. It is unique for a document in a project.

13.6.3 Table of Contents

Document

Classes and functions for rendering documents.

`class disseminate.document.document.Document(`*src_filepath*, *environment*, *parent_context=None*, *level=1*`)`

Bases: `object`

A base class document rendered from a source file to one or more target files.

Parameters

src_filepath [Union[*SourcePath*, str]] The path (an absolute path or a path relative to the current directory) for the document (markup source) file of this document.

target_root [Optional[Union[*TargetPath*, str]]] The path for the rendered target files. Subdirectories for the targets will be created. (ex: 'html' 'tex') By default, if not specified, the target_root will be one directory above the project_root.

parent_context [Optional[*DocumentContext*]] The context of the parent document or default context.

level [Optional[int]] The level of document loaded. Sub-documents are loaded at a higher level (>1) than the root document.

Attributes

src_filepath [*SourcePath*] The filename and path for this document's (markup source) file. This file should exist. ex: 'src/chapter1/chapter1.dm'

context [*DocumentContext*] A context dict with the values needed to render a target document.

subdocuments [OrderedDict[*SourcePath*, *Document*]] An ordered dict with the sub-documents included in this document. The keys are src_filepaths, and the values are the sub-documents themselves. The documents are ordered according to their placement in the document tree (project).

This document owns the sub-documents and only weak references to these documents should be made to these documents. (i.e. when the sub-documents dict is cleared, the memory for the document objects should be released)

build(*complete=True*)
Run a build of the document and all subdocuments.

build_needed()
Evaluate whether a build is required

property doc_id
The unique string identifier (within a project) for the document.

property doc_ids
The list of all doc_id for this document and all subdocuments.

documents_by_id(*document=None*, *only_subdocuments=False*, *recursive=True*)
Produce an ordered dict of a document and sub-documents entered by doc_id.

Parameters

document [Optional[*Document*]] The document for which to create the document dict. If None is specified, this document will be used.

only_subdocuments [Optional[bool]] If True, only the sub-documents will be returned (not this document or the document specified.)

recursive [Optional[bool]] If True, the sub-documents of sub-documents are returned (in order) in the ordered dict as well

Returns

document_dict [OrderedDict[str, *Document*]] An ordered dict of documents. The keys are doc_ids and the values are document objects.

documents_dict(*document=None*, *only_subdocuments=False*, *recursive=False*)
Produce an ordered dict of a document and all its sub-documents.

Parameters

document [Optional[*Document*]] The document for which to create the document list. If None is specified, this document will be used.

only_subdocuments [Optional[bool]] If True, only the sub-documents will be returned (not this document or the document specified.)

recursive [Optional[bool]] If True, the sub-documents of sub-documents are returned (in order) in the ordered dict as well

Returns

document_dict [OrderedDict[*SourcePath*, *Document*]] An ordered dict of documents. The keys are src_filepaths and the values are document objects.

documents_list(*document=None*, *only_subdocuments=False*, *recursive=False*)
Produce an ordered list of a document and all its sub-documents.

Parameters

document [Optional[*Document*]] The document for which to create the document list.

only_subdocuments [Optional[bool]] If True, only the sub-documents will be returned (not this root document or the document specified.

recursive [Optional[bool]] If True, the sub-documents of sub-documents are returned (in order) in the list as well

Returns

document_list [List[*Document*]] An ordered list of documents.

load(*reload=False*, *level=1*)
Load or reload the document into the context.

Parameters

reload [Optional[bool]] If True, force the reload of the document.

level [Optional[int]] The level of document loaded. Sub-documents are loaded at a higher level (>1) than the root document.

Returns

document_loaded [bool] True, if a sub-document was (re)loaded.

load_required()
Evaluate whether a load is required.

Returns

load_required [bool] True, if a load is required. False if a load isn't required.

load_subdocuments(*level=1*)
Load the sub-documents listed in the include entries in the context.

Parameters

level [Optional[int]] The level of document loaded. Sub-documents are loaded at a higher level (>1) than the root document.

Returns

document_loaded [bool] True, if a sub-document was (re)loaded.

property short
The short title for the document.

target_filepath(*target*)
The filepath for the given target extension.

Parameters

target [str] The target extension for the target file. ex: '.html' or '.tex'

Returns

target_filepath [Union[*TargetPath*, None]] The target filepath.

property targets
The targets dict.

There may be more targets in this dict than those listed in the context because some targets may be intermediary targets in a cache path.

Returns

targets [Dict[str, *TargetPath*]] The targets are a dict with the target extension as keys (ex: '.html') and the value is the target_filepath for that target. (ex: 'html/index.html') These paths are target paths.

property title
The title for the document.

Document Context

The document context

class disseminate.document.document_context.**DocumentContext**(*document*, **args*, ***kwargs*)
Bases: *disseminate.context.context.BaseContext*

A context dict used for documents objects.

do_not_inherit = {'body', 'builders', 'doc_id', 'document', 'include', 'includes', 'mtime', 'paths', 'short'}
The keys for context entries that should not be inherited from the parent context. They should be unique to each context.

property document
The document that owns this context

exclude_from_reset = {'document', 'environment', 'label_manager', 'paths'}
The keys for context entries that should not be removed when the context is cleared. These are typically for entries setup in the __init__ of the class.

property includes
Retrieve a list of included subdocument source paths from the 'include' entry of the context.

Returns

include_list [List[*SourcePath*]] A list of the paths for the included subdocuments.

reset()
(Selectively) resets the context to its initial state.

The context is reset by removing items with keys not specified in the 'exclude_from_clear' class attribute.

Note: Entries in the parent_context are copied to this context. If the parent_context entry is a mutable, like a list or dict, then a copy of that mutable is created if a 'copy' method exists for that mutable. However, mutables within those mutables will still point to the original.

Examples

```
>>> l1, l2 = ['a'], ['b']
>>> context = BaseContext(l1=l1)
>>> context['l2'] = l2
>>> context['l1'].append(1)
>>> context['l2'].append(2)
>>> context.reset()
>>> context['l1'] == ['a']  # reset to original l1
True
>>> 'l2' in context
False
```

property root_document
The root document for a project

target_filepath(*target*)
Return the target filepath for this document's target, given the target.

The target filepaths are retrieved from the builders. Accordingly, these should already have been loaded in the context. Additionally, this function may return target_filepaths for targets not listed by the 'targets' property. This is because these targets may be intermediary targets stored in a cache directory.

Parameters

target [str] The target to search

Returns

target_filepath [*paths.TargetPath*] The filepath for the target file of the document.

Raises

TargetNotFound Raised if a target was requested but not found.

target_filepaths()
Return all target filepaths for this document's targets

The target filepaths are retrieved from the builders. Accordingly, these should already have been loaded in the context. Additionally, this function may return target_filepaths for targets not listed by the 'targets' property. This is because these targets may be intermediary targets stored in a cache directory.

Returns

targets [Dict[str, *paths.TargetPath*]] The targets dict. The strings are the targets and the values are the corresponding target_filepaths.

property targets
Retrieve a list of targets from the 'targets' or 'target' entry of the document.

Returns

target_list [List[str]] A list of targets specified in the context.

validation_types = {'doc_id': <class 'str'>, 'document': None, 'environment': None, 'inactive_tags': <class 'set'>, 'label_fmts': <class 'dict'>, 'label_manager': <class 'disseminate.label_manager.label_manager.LabelManager'>, 'label_resets': <class 'dict'>, 'mtime': <class 'float'>, 'paths': <class 'list'>, 'process_context_tags': <class 'set'>, 'process_paragraphs': <class 'set'>, 'project_root': <class 'disseminate.paths.paths.SourcePath'>, 'src_filepath': <class 'disseminate.paths.paths.SourcePath'>, 'target_root': <class 'disseminate.paths.paths.TargetPath'>, 'targets': <class 'set'>}

Required entries in the document context dict to be a valid document context–as well as their matching type to be checked.

Exceptions

Exceptions for documents

exception disseminate.document.exceptions.**DocumentException**
Bases: Exception

An error was encountered in creating, setting up or rendering a document.

exception disseminate.document.exceptions.**TargetNotFound**
Bases: *disseminate.document.exceptions.DocumentException*

An error raised if a document target was requested but not found.

Signals

Signals for document events

disseminate.document.signals.**document_created = <disseminate.signals.signals.Signal object>**
Signal sent after document creation. Receivers take a document parameter.

disseminate.document.signals.**document_deleted = <disseminate.signals.signals.Signal object>**
Signal sent before a document is deleted. Receivers take a document parameter.

disseminate.document.signals.**document_onload = <disseminate.signals.signals.Signal object>**
Signal sent when a document is loaded. Receivers take a document or document context parameter.

disseminate.document.signals.**document_build = <disseminate.signals.signals.Signal object>**
Signal sent when a document's targets are being built to their final target files. Receivers take a document parameter.

disseminate.document.signals.**document_build_needed = <disseminate.signals.signals.Signal object>**
Signal sent to evaluate whether a build is needed. Takes a document as a parameter and returns True or False

disseminate.document.signals.**document_tree_updated = <disseminate.signals.signals.Signal object>**
Signal sent when a root document or one of its sub-documents was re-loaded. Takes a root document as a parameter.

Receivers

disseminate.document.receivers.**delete_document**(*document*, ***kwargs*)
Reset the context and managers for a document on document deletion.

disseminate.document.receivers.**load_document**(*document*, ***kwargs*)
Load the document text file into the document context.

disseminate.document.receivers.**process_document_label**(*context*, ***kwargs*)
A context processor to set the document label in the label manager.

disseminate.document.receivers.**process_headers**(*context*, ***kwargs*)
Process header strings for entries in a context by loading them into the context.

Context information comes from the following sources:

1. The document itself and its header file
2. Additional headers from the template

The information in (1) takes precedence over (2). For this reason, (1) should be loaded after (2), but (2) is specified by some entries, like renderers, template, targets, in (1), so (1) is pre-loaded before (2).

disseminate.document.receivers.**process_tags**(*context*, ***kwargs*)
Convert context entries into tags for entries listed the process_context_tags' context entry.

This function converts tags for context entries identified by keys listed in the 'process_context_tags' context entry

---

Note: This function is designed to work with string macro entries. These are identified by context entries with keys that start with the settings.tag_prefix (e.g. '@test'). These *should not* be converted into asts, as they are required for simple string replacement.

---

disseminate.document.receivers.**reset_document**(*document*, ***kwargs*)
Reset the context and managers for a document on load.

13.7 Formats

Wrappers and utilities for formatting text and tags into different formats

13.7.1 XHTML Format

Utilities for formatting html strings and text.

exception disseminate.formats.xhtml.**XHtmlFormatError**
Bases: *disseminate.formats.exceptions.FormattingError*

Error in xhtml formatting.

disseminate.formats.xhtml.**xhtml_entity**(*entity*, *level=1*, *method='html'*, *pretty_print=True*)
Format an html entity string.

Parameters

entity [str] an html entity string

level [Optional[str]] The level of the tag.

method [Optional[str]] The rendering method. 'html', 'xhtml' or 'xml'

pretty_print [Optional[bool]] If True, make the formatted html pretty–i.e. with newlines and spacing for nested tags.

Returns

xhtml [str] The entity formatted in xhtml.

Raises

XHtmlFormatError Raised if the contents of the tag aren't a simple string. i.e. nested tags are not allowed.

Examples

```
>>> xhtml_entity('alpha')
Markup('&alpha;\n')
```

disseminate.formats.xhtml.**xhtml_list**(**elements*, *attributes=None*, *listtype='ol'*, *level=1*, *target=None*, *method='html'*, *pretty_print=True*, *inner=False*)

A xhtml list element.

Parameters

elements [Tuple[Tuple[int, `lxml.builder.E`]]] Each element is a tuple of the list element level and the 'li' lxml element.

attributes [Optional[Union[*Attributes*, str]]] The attributes of the tag.

listtype [Optional[str]] The type of list to create. ex: ul, ol

level [Optional[int]] The level of the tag.

target [Optional[str]] If speficied, filter the attributes that match the given target.

method [Optional[str]] The rendering method. 'html', 'xhtml' or 'xml'

pretty_print [Optional[bool]] If True, make the formatted html pretty–i.e. with newlines and spacing for nested tags.

inner [Optional[bool]] If True, this function is invoked as an inner html list. This is useful for adding html attributes only to the outer list.

Returns

xhtml [str] If level=1, a string formatted in html if level>1, an html element (`lxml.build.E`)

Raises

XHtmlFormatError [`HtmlFormatError`] A TagError is raised if a non-allowed list environment is used

disseminate.formats.xhtml.**xhtml_tag**(*name*, *attributes=None*, *formatted_content=None*, *level=1*, *target=None*, *method='html'*, *nsmap=None*, *pretty_print=True*)

Format an xhtml tag string.

Parameters

name [Optional[str]] The name of the html tag.

attributes [Optional[Union[*Attributes*, str]]] The attributes of the tag.

formatted_content [Optional[Union[str, list, `lxml.builder.E`]]] The contents of the html tag.

level [Optional[int]] The level of the tag.

target [Optional[str]] If speficied, filter the attributes that match the given target.

method [Optional[str]] The rendering method. 'html', 'xhtml' or 'xml'

nsmap [Optional[dict]] Optional namespace map to create the tag with.

pretty_print [Optional[bool]] If True, make the formatted html pretty–i.e. with newlines and spacing for nested tags.

Returns

html [str] If level=1, a string formatted in (x)html if level>1, an html element (`lxml.build.E`)

Raises

XHtmlFormatError [*XHtmlFormatError*] A TagError is raised if a non-allowed environment is used

Examples

```
>>> xhtml_tag('img', attributes="src='test.svg'")
Markup('<img src="test.svg">\n')
>>> xhtml_tag('img', attributes="src='test.svg'", method='xml')
Markup('<img src="test.svg"/>\n')
```

13.7.2 LaTeX Format

Utilities for formatting tex strings and text.

exception disseminate.formats.tex.**TexFormatError**

Bases: *disseminate.formats.exceptions.FormattingError*

Error in latex formatting.

disseminate.formats.tex.**tex_cmd**(*cmd*, *attributes=''*, *formatted_content=None*, *indent=None*)

Format a tex command.

Parameters

cmd [Optional[str]] The name of the LaTeX command to format.

attributes [Optional[Union[*Attributes*, str]]] The attributes of the tag.

formatted_content [Optional[str]] The contents of the tex environment formatted as a string in LaTeX. If not specified, the tex_str will not be used as a LaTeX parameter

indent [Optional[int]] If specified, indent lines by the given number of spaces.

Returns

tex_env [str] The LaTeX environment string

Raises

TexFormatError [*TexFormatError*] A TexFormatError is raised if an non-allowed environment is used.

disseminate.formats.tex.**tex_env**(*env*, *attributes*, *formatted_content*, *min_newlines=False*, *indent=None*)

Format a tex environment.

Parameters

env [str] The name of the LaTeX environment to format.

attributes [Union[*Attributes*, str]] The attributes of the tag.

formatted_content [str] The contents of the tex environment formatted as a string in LaTeX.

min_newlines [Optional[bool]] If True, extra new lines before, after and in the environment will not be included.

indent [Optional[int]] If specified, indent lines by the given number of spaces.

Returns

tex_env [str] The LaTeX environment string

Raises

TexFormatError [*TexFormatError*] A TexFormatError is raised if an non-allowed environment is used.

disseminate.formats.tex.**tex_verb**(*formatted_content*)
Format a tex verb command

Parameters

formatted_content [str] The contents of the tex environment formatted as a string in LaTeX.

Returns

tex_verb [str] The LaTeX verb string

13.7.3 Exceptions

exception disseminate.formats.exceptions.**FormattingError**
Bases: Exception

An error in converting to a specific format.

13.8 Label Manager

A manager for labels from headings, figures, equations, tables and other document components.

13.8.1 Responsibilities

The labels and label manager are responsible for:

1. *Numbering*. Accurate and consistent formatting of labels and their identifer. *e.g.* Fig. 1.
2. *Internal links*. Work in conjunction with the `@ref` to correctly link to these elements.
3. *Tracking modification times*. Keep track of the modification times for the label so that references to those labels can trigger a document rendering.

13.8.2 Label Manager

The manager for labels.

class disseminate.label_manager.label_manager.**LabelManager**(*root_context*)
Bases: object

Manage labels for a project (a document tree).

Parameters

root_context [*DocumentContext*] The context for the root document. The label manager does not own the context object, so only a weak reference to the context is stored.

Attributes

labels [Dict[Tuple[str,str], Label]] A list of labels where the key is the (doc_id, label_id) and the values are the label objects.

add_content_label(*id*, *kind*, *title*, *context*)
Add content label.

See *add_label()* for usage details.

add_document_label(*id*, *kind*, *title*, *context*)
Add document label.

See *add_label()* for usage details.

add_label(*id*, *kind*, *context*, *label_cls*, **args*, ***kwargs*)
Add a label.

Parameters

id [str] The unique identifier (within a project of documents) for the label. The id includes the label_id and possibly the doc_id of the label. 'test.dm::intro'. (See parse_id for details) ex: 'ch:nmr-introduction'

kind [Union[str, List[str], Tuple[str]]] The kind of the label. ex: 'figure', ('heading', 'chapter'), 'equation'

context [*DocumentContext*] The context for the document adding the label. (This may be different from the context of the root document, self.root_context)

label_cls [Type[Label]] The label class (or subclass) to use in creating the label.

Raises

DuplicateLabel [*DuplicateLabel*] Raised if a label with the same id already exists in this label manager.

format_string(*id*, **keys*, *target=None*)
Retrieve the formatted label string for a label.

Label format strings are intended to be replaced with the values of the label by replace_macros (replace_macros).

Parameters

id [str] The label of the label ex: 'ch:nmr-introduction'

keys [Optional[List[str], Tuple[str]]] If specified, use the given keys to find entries in the format_str dict. (see *find_entry*)

target [Optional[str]] If specified, try finding format strings for the given target.

get_label(*id*, *register=True*, *context=None*)
Return the label for the given label id.

Note: This function registers the added labels.

Parameters

id [str] The unique identifier (within a project of documents) for the label. The id includes the label_id and possibly the doc_id of the label. 'test.dm::intro'. (See parse_id for details) ex: 'ch:nmr-introduction'

register [Optional[bool]] If True, labels will be registered before doing the search

context [*DocumentContext*] The context for the document adding the label. (This may be different from the context of the root document, self.root_context)

Returns

label [Type[Label]] The corresponding label.

Raises

LabelNotFound [*LabelNotFound*] A LabelNotFound exception is raised if a label with the given id could not be found.

get_labels_by_id(*ids*, *register=True*, *context=None*)
Return the labels with the given (optional) doc_ids and label_ids.

Parameters

ids [Union[str, Tuple[str]]] The unique identifiers (within a project of documents) for the label. The id includes the label_id and possibly the doc_id of the label. 'test.dm::intro'. (See parse_id for details) ex: 'ch:nmr-introduction'

register [Optional[bool]] If True, labels will be registered before doing the search

context [*DocumentContext*] The context for the document adding the label. (This may be different from the context of the root document, self.root_context)

Returns

labels [List[Type[Label]]] A list of label objects.

Raises

LabelNotFound [*LabelNotFound*] A LabelNotFound exception is raised if a label with the given id could not be found.

get_labels_by_kind(*doc_id=None*, *kinds=None*, *register=True*)
Return a filtered and sorted list of all labels for the given document and kinds.

Note: This function registers the added labels.

Parameters

doc_id [Optional[str]] If specified, only label for the given document id will be returned. (This is used an alternative to the context.)

kinds [Optional[Union[str, List[str], Tuple[str]]] If None, all label kinds are returned. If string, all labels matching the kind string will be returned. If a list of strings is returned, all labels matching all the kinds listed will be returned.

register [Optional[bool]] If True, labels will be registered before doing the search

Returns

labels [List[Type[Label]]] A list of label objects.

register(*context=None*)
Register the labels.

reset(*doc_ids=None*)
Reset the the labels.

Parameters

doc_ids [Union[str, List[str], Tuple[str]]] If specified, remove the labels for the given doc_ids

disseminate.label_manager.label_manager.**parse_id**(*id*, *context=None*)
Parses a label's identifier into a doc_id and label_id.

Parameters

id [str] The unique identifier (within a project of documents) for the label. The id includes the label_id, and it might include the doc_id. ex: 'test.dm::intro'. (See label_sep for details) 'ch:nmr-introduction'

context [*DocumentContext*] The document context.

Returns

doc_id, label_id [Tuple[Union[str, None], str]] The doc_id and label_id from the label id.

Examples

```
>>> parse_id('intro')
(None, 'intro')
>>> parse_id('fig:diagram-1')
(None, 'fig:diagram-1')
>>> parse_id('test1.dm::fig:diagram-1')
('test1.dm', 'fig:diagram-1')
```

13.8.3 Types

Label

The label base class for the label manager.

class disseminate.label_manager.types.label.**Label**(*doc_id*, *id*, *kind*, *order=None*)
Bases: object

A label used for referencing.

Parameters

doc_id [str] The unique identifier (for documents within a label manager) for the document that owns the label.

id [str] The unique identifier(s) of the label. ex: 'nmr_introduction'. This should be unique for the entire project.

kind [Tuple[str]] The kind of the label is a tuple that identifies the kind of a label from least specific to most specific. ex: ('figure',), ('chapter',), ('equation',), ('heading', 'chapter',)

order [Optional[Tuple[int]]] The order/number of the label. The order is a tuple of integers with a length that matches the 'kind' tuple. Each entry represents the order/number of each kind ex: for a kind ('heading', 'chapter') could have an order of (3, 2) which would represent the 3rd 'heading' and 2nd 'chapter' item. Some of the orders may be reset, but the first item should not–e.g. the 'heading' count should represent the running count of all headings, while the chapter count may be reset. This ensures that the order of labels is preserved when the counter of sub-kinds are reset. (see `OrderLabels`)

property number
The number for the label's most specific kind.

ContentLabel

class disseminate.label_manager.types.content_label.**ContentLabel**(*doc_id*, *id*, *kind*, *title*, *order=None*)

Bases: *disseminate.label_manager.types.label.Label*

A label for content, like a heading (chapter, section, subsection) or an item that should show up in a table of contents, like a figure caption.

Attributes

document_label [DocumentLabel] The label object for the document that owns this label.

chapter_label [Optional[ContentLabel]] The label for the chapter under which this label is under.

section_label [Optional[ContentLabel]] The label for the section under which this label is under.

subsection_label [Optional[ContentLabel]] The label for the subsection under which this label is under.

subsubsection_label [Optional[ContentLabel]] The label for the subsubsection under which this label is under.

property chapter_number
The number for the chapter to which this label belongs.

property chapter_title
The title for the chapter to which this label belongs.

property part_number
The number for the part to which this label belongs.

property part_title
The title for the part to which this label belongs.

property section_number
The number for the section to which this label belongs.

property section_title
The title for the section to which this label belongs.

property subsection_number
The number for the subsection to which this label belongs.

property subsection_title
The title for the subsection to which this label belongs.

property subsubsection_number
The number for the subsubsection to which this label belongs.

property subsubsection_title
The title for the subsubsection to which this label belongs.

property tree_number
The string for the number for the chapter, section, subsection and so on.

e.g. Section 3.2.1.

DocumentLabel

class disseminate.label_manager.types.document_label.**DocumentLabel**(*doc_id*, *id*, *kind*, *title*, *order=None*)

Bases: *disseminate.label_manager.types.label.Label*

A label for documents.

13.8.4 Receivers

Signals for label events

disseminate.label_manager.receivers.**reset_label_manager**(*context*, ***kwargs*)

Reset the label manager in the context on document load.

This should be done before the text of the document is loaded in the document object.

13.8.5 Signals

Signals for processing labels.

disseminate.label_manager.signals.**label_register = <disseminate.signals.signals.Signal object>**

A signal sent when labels are to be registered. Receivers take registered labels (a list of labels) and collected labels (dict of labels)

13.8.6 Label Exceptions

Exceptions for labels

exception disseminate.label_manager.exceptions.**DuplicateLabel**

Bases: *disseminate.label_manager.exceptions.LabelError*

A label that was already defined is defined again

exception disseminate.label_manager.exceptions.**LabelError**

Bases: Exception

An error was encountered while processing a label.

exception disseminate.label_manager.exceptions.**LabelNotFound**

Bases: *disseminate.label_manager.exceptions.LabelError*

Could not find a reference to a label

13.9 Paths

Path objects (pathlib.Path) that can keep track of subpaths for a project and targets.

13.9.1 Paths

Classes for different kinds of paths.

class disseminate.paths.paths.**SourcePath**(*project_root=''*, *subpath=''*)

Bases: object

A path for a file in the source directory that keeps track of the project_root and subpath.

Parameters

project_root [Optional[str]] The path for the project's root directory.

subpath [Optional[str]] The relative path within the project's root directory.

use_name(*name*)

with_name(name) that returns a SourcePath.

A new method name is used since the flavor_cls method takes precedence.

Examples

```
>>> p = SourcePath(project_root='/media', subpath='tests/fig1.png')
>>> p.use_name('fig2.png')
SourcePath('/media/tests/fig2.png')
>>> p.use_name('fig2.png').subpath
SourcePath('tests/fig2.png')
```

use_subpath(*subpath*)

Returns a SourcePath with the subpath replaced with the given string or path.

Examples

```
>>> p = SourcePath(project_root='/media', subpath='tests/fig1.png')
>>> p.use_subpath('text.txt')
SourcePath('/media/text.txt')
>>> p.use_subpath('text.txt').subpath
SourcePath('text.txt')
```

use_suffix(*suffix*)

with_suffix(suffix) that returns a SourcePath.

A new method name is used since the flavor_cls method takes precedence.

Examples

```
>>> p = SourcePath(project_root='/media', subpath='tests/fig1.png')
>>> p.use_suffix('.pdf')
SourcePath('/media/tests/fig1.pdf')
>>> p.use_name('fig1.pdf').subpath
SourcePath('tests/fig1.pdf')
```

class disseminate.paths.paths.**TargetPath**(*target_root=''*, *target=''*, *subpath=''*)

Bases: object

A path for a file in a target directory that keeps track of the target_root, target and subpath.

Parameters

target_root [Optional[str]] The path for the root directory for rendered files.

target [Optional[str]] The sub-directory for the target type to render. ex: 'html' or 'tex'

subpath [Optional[str]] The relative path within the target sub-directory.

get_url(*context=None*, *target=None*)
Construct the url for the path.

use_name(*name*)
with_name(name) that returns a TargetPath.

A new method name is used since the flavor_cls method takes precedence.

Examples

```
>>> p = TargetPath(target_root='/media', subpath='tests/fig1.png')
>>> p.use_name('fig2.png')
TargetPath('/media/tests/fig2.png')
>>> p.use_name('fig2.png').subpath
TargetPath('tests/fig2.png')
```

use_subpath(*subpath*)
Returns a TargetPath with the subpath replaced with the given string or path.

Examples

```
>>> p = TargetPath(target_root='/media', subpath='tests/fig1.png')
>>> p.use_subpath('text.txt')
TargetPath('/media/text.txt')
>>> p.use_subpath('text.txt').subpath
TargetPath('text.txt')
```

use_suffix(*suffix*)
with_suffix(suffix) that returns a TargetPath.

A new method name is used since the flavor_cls method takes precedence.

Examples

```
>>> p = TargetPath(target_root='/media', subpath='tests/fig1.png')
>>> p.use_suffix('.pdf')
TargetPath('/media/tests/fig1.pdf')
>>> p.use_suffix('.pdf').subpath
TargetPath('tests/fig1.pdf')
```

13.9.2 Utils

Utilities for paths.

disseminate.paths.utils.**find_file**(*path*, *context*, *raise_error=True*)

Search for an existing file given the path and, if needed, the 'paths' entry from a context.

Parameters

path [Union[str, `pathlib.Path`]] The path stub to search.

context [`context.Context`] The document's context with an entry of 'paths' to search.

raise_error [Optional[bool]] If True (default), raise error if a file isn't found.

Returns

valid_path [`pathlib.Path`] A path for a file that exists.

Raises

FileNotFoundError Raised if the file could not be found.

disseminate.paths.utils.**find_files**(*string*, *context*)

Determine if filenames for valid files are present in the given string.

Parameters

string [str] The string that may contain filenames

context [*DocumentContext*] A document context that contains paths to search.

Returns

filepaths [List[`pathlib.Path`]] List of found paths

disseminate.paths.utils.**rename**(*path*, *filename=None*, *append=None*, *extension=None*)

Rename the filename from a path with optional modifiers.

Parameters

path [Union[str, `pathlib.Path`]] The path to rename

filename [Optional[str]] The new filename to rename the filename (without the extension)

append [Optional[str]] If specified, add the string to the filename

extension [Optional[str]] If specified, replace the extension with the given extension

Returns

new_path [`pathlib.Path`] The new path with the filename renamed.

Examples

```
>>> rename('/tmp/ch1_file.png', filename='test')
PosixPath('/tmp/test.png')
>>> rename('/tmp/ch1_file.png', append='_crop2')
PosixPath('/tmp/ch1_file_crop2.png')
>>> rename('/tmp/ch1_file.png', append='_crop2', extension='ext')
PosixPath('/tmp/ch1_file_crop2.ext')
>>> rename('/tmp/ch1.1_file.png', filename='test')
PosixPath('/tmp/test.png')
>>> rename('/tmp/ch1.1_file.png', filename='test', append='_crop2')
```

(continues on next page)

(continued from previous page)

```
PosixPath('/tmp/test_crop2.png')
>>> rename('/tmp/ch1.1_file.png', filename='test', append='_crop2',
...        extension='svg')
PosixPath('/tmp/test_crop2.svg')
>>> rename('/tmp/ch1.1_file', extension='html')  # this won't work
PosixPath('/tmp/ch1.html')
```

13.10 Server

13.10.1 App

The Tornado main app

class disseminate.server.app.**TornadoApp**(***kwargs*)
Bases: tornado.web.Application

The Tornado App for the built-in webserver

disseminate.server.app.**get_app**(*in_path*, *out_dir*, *debug=False*, ***kwargs*)
Create the Tornado app instance

disseminate.server.app.**run_server**(*in_path*, *out_dir*, *port=8899*, *debug=False*)
Create and run the web-server.

Parameters

in_path [str] The filenames for project root documents.

out_dir [str] The target root directory to render files to.

port [Optional[int]] The network port to serve the web-server.

debug [Optional[bool]] If true, include debugging information.

13.10.2 URL

URL routes for the Tornado server

13.10.3 Handlers

class disseminate.server.handlers.**CheckerHandler**(*application: tornado.web.Application*, *request: tornado.httputil.HTTPServerRequest*, ***kwargs: Any*)
Bases: *disseminate.server.handlers.server.ServerHandler*

A request handler for the dependency checkers

checker_to_dict(*item*, *required=True*, *level=1*)
Convert checkers into a list of dicts suitable for rendering the view.

class disseminate.server.handlers.**CustomStaticFileHandler**(*application: tornado.web.Application*, *request: tornado.httputil.HTTPServerRequest*, ***kwargs: Any*)
Bases: *disseminate.server.handlers.server.ServerHandler*, tornado.web.StaticFileHandler

A custom static file handler that handles rendered error pages

class disseminate.server.handlers.**PygmentizeHandler**(*application: tornado.web.Application*, *request: tornado.httputil.HTTPServerRequest*, ***kwargs: Any*)

Bases: *disseminate.server.handlers.server.ServerHandler*

A handler to pygmentize source files

class disseminate.server.handlers.**ServerHandler**(*application: tornado.web.Application*, *request: tornado.httputil.HTTPServerRequest*, ***kwargs: Any*)

Bases: tornado.web.RequestHandler

A request handler for server functions

get_template_path()
The template path for the server templates

set_default_headers()
Override this to set HTTP headers at the beginning of the request.

For example, this is the place to set a custom `Server` header. Note that setting such headers in the normal flow of request processing may not do what you want, since headers may be reset during error handling.

write_error(*status_code*, ***kwargs*)
Render custom error pages

class disseminate.server.handlers.**SignalHandler**(*application: tornado.web.Application*, *request: tornado.httputil.HTTPServerRequest*, ***kwargs: Any*)

Bases: *disseminate.server.handlers.server.ServerHandler*

A request handler for the signals

signals_to_dict(*signal_namespace*)
Given a list of processor classes, prepare a list of dicts listing the sub-processors Parameters ———- signal_namespace : *signals.signals.Namespace*

class disseminate.server.handlers.**TreeHandler**(*application: tornado.web.Application*, *request: tornado.httputil.HTTPServerRequest*, ***kwargs: Any*)

Bases: *disseminate.server.handlers.server.ServerHandler*

A request handler for the project document tree

tree_to_dict(*docs*, *level=1*)
Convert the root documents into a list of dicts, suitable for rendering in the view.

13.11 Signals

13.11.1 Signals

class disseminate.signals.signals.**Namespace**

Bases: dict

A mapping of signal names to signals.

signal(*name*, *doc=None*)
Return a signal with the given name.

Parameters

name [str] The name of the signal

doc [Optional[str]] The description documentation for the signal.

class disseminate.signals.signals.**Signal**(*name*, *doc=None*)

Bases: object

A notification emitter.

(Inspired by blinker)

Attributes

receivers [Dict[int, Callable]] A dict with the order (key) for a receiver (value) to run when the signal is emitted.

connect(*receiver*, *order*, *weak=True*)

Connect a receiver to this signal.

Parameters

receiver [Callable] A function that will be executed when this signal is emitted.

order [int] The order for this receiver to be run.

weak [Optional[bool]] If true, a weak reference will be stored for the receiver.

connect_via(*order*, *weak=True*)

The decorator for connect

emit(***kwargs*)

Emit (send) the signal and run the receiver functions.

receivers_dict()

Return a dict of receivers (values) and their orders (keys).

reset()

Reset the signal to its initial state

13.11.2 Exceptions

exception disseminate.signals.exceptions.**DuplicateSignal**

Bases: Exception

Raised when two receivers with the same order are assigned.

13.12 Tags

Tags are interpreted text elements in a disseminate document that are converted to other elements in the rendered document, like bold text, images, figures, equations and tables.

13.12.1 Tag

Core classes and functions for tags.

class disseminate.tags.tag.**Tag**(*name*, *content*, *attributes*, *context*)

A tag to format text in the markup document.

Parameters

name [str] The name of the tag as used in the disseminate source. (ex: 'body')

content [Union[str, List[Union[str, list, *Tag*], *Tag*]] The contents of the tag. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Union[str, *Attributes*]] The attributes of the tag. These are distinct from the Tag class attributes; they're the customization attributes specified by the user to change the appearance of a rendered tag. However, some or all attributes may be used as tag attributes, depending on the settings.html_valid_attributes, settings.tex_valid_attributes and so on.

context [*Type[BaseContext]*] The context with values for the document. The tag holds a weak reference to the context, as it doesn't own the context.

Attributes

aliases [Tuple[str]] A tuple of strings for other names a tag goes by

hash [Optional[str]] The hash for the tag's contents before processing. This is useful for detecting changes in the string's contents.

html_name [str] If specified, use this name for the html tag. Otherwise, use name.

tex_cmd [str] If specified, use this name to render the tex command.

tex_env [str] If specified, use this name to render the tex environment.

tex_paragraph_newlines [bool] If True (default), block tags within paragraphs will be typeset with a newline before and after. This is disable, for example, for equations.

active [bool] If True, the Tag can be used by the TagFactory.

process_content [bool] If True, the contents of the tag will be parsed and processed into a tag tree (AST) by the ProcessContent processor on creation.

process_typography [bool] If True, the strings in contents of the tag will be parsed and processed for typography characters by the ProcessTypography processor on creation.

include_paragraphs [bool] If True, then the contents of this tag can be included in paragraphs. See `disseminate.tags.processors.process_paragraphs()`.

paragraph_role [str] If this tag is directly within a paragraph, the type of paragraph may be specified with a string. The following options are possible:

- None : The paragraph type has not been assigned
- 'inline' : The tag is within a paragraph that includes a mix of strings and tags
- 'block' : The tag is within its own paragraph.

copy(*new_context=None*)

Create a copy of this tag and all sub-tabs.

The tag copy is a deep copy of the attributes and content, but the context is either kept the same or replaced with the specified new_context.

Parameters

new_context [Optional[*Type[BaseContext]*]] The new context to replace, if specified.

default_fmt(*content=None*, *attributes=None*)
Convert the tag to a text string.

Strips the tag information and simply return the content of the tag.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

Returns

text_string [str] A text string with the tags stripped.

flatten(*tag=None*, *filter_tags=True*)
Generate a flat list with this tag and all sub-tags and elements.

Parameters

tag [Optional[*Tag*]] If specified, flatten the given tag, instead of this tag.

filter_tags [Optional[bool]] If True, only return a list of tag objects. Otherwise, include all content items, including strings and lists.

Returns

flattened_list [List[Union[str, *Tag*]]] The flattened list.

html_fmt(*content=None*, *attributes=None*, *format_func='html_fmt'*, *method='html'*, *level=1*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

property short
The short title for the tag

tex_fmt(*content=None*, *attributes=None*, *mathmode=False*, *level=1*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

property title
A title string converted from the content

xhtml_fmt(*method='xhtml'*, *format_func='xhtml_fmt'*, ***kwargs*)
Convert the tag to an xhtml string or html element.

Returns

xhtml [str or xhtml element] A string in XHTML format or an XHTML element (`lxml.builder.E`).

class disseminate.tags.tag.**TagFactory**
Generates the appropriate tag for a given tag type.

The tag factory instantiates tags based on loaded modules and initialization parameters.

Parameters

tag_base_cls [*Tag*] The base class for Tag objects.

classmethod tag(*tag_name*, *tag_content*, *tag_attributes*, *context*)
Return the approriate tag, given a tag_name and tag_content.

A tag subclass, rather than the Tag base class, will be returned if

- A tag subclass with the tag_name (or with an alias) is available.
- The tag subclass has an 'active' attribute that is True
- The tag's name isn't listed in the 'inactive_tags' set in the context.

Parameters

tag_name [str] The name of the tag. ex: 'bold', 'b'

tag_content [Union[str, list]] The content of the tag. ex: 'this is bold'

tag_attributes [str] The attributes of a tag. ex: 'width=32pt'

context [`document.DocumentContext`] The document's context.

Returns

tag [`Tag`] An instance of a Tag subclass.

classmethod tag_class(*tag_name*, *context*)
Retrieve the tag class for the given tag_name

classmethod tag_classes()
A dict of all the active Tag subclasses.

13.12.2 Asy

Tags to render asymptote (asy) figures and diagrams

class disseminate.tags.asy.**Asy**(**args*, ***kwargs*)

Bases: *disseminate.tags.img.Img*

The asy tag for inserting asymptote images.

13.12.3 Caption

Tags for figure captions and references.

class disseminate.tags.caption.**Caption**(**args*, ***kwargs*)

Bases: *disseminate.tags.tag.Tag*, *disseminate.tags.label.LabelMixin*

A tag for captions.

Note: The use of a naked caption tag is allowed (i.e. a caption not nested within a figure or table), but this won't register a label with the label manager. In order to create a label, the create_label method in the LabelMixin must be invoked.

example

```
@fig{@img{image.svg}
     @caption{My first figure}
    }
```

---

default_fmt(*attributes=None*, *content=None*)

Convert the tag to a text string.

Strips the tag information and simply return the content of the tag.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

Returns

text_string [str] A text string with the tags stripped.

generate_label_id()

Generate the label id and set the id in the attributes.

generate_label_kind()

Generate the kind tuple for the created label.

Override this function to customize the generation of the label kind.

Returns

kind [Tuple[str]] A tuple of strings for the kind of label. ex: ('heading', 'chapter')

html_fmt(*content=None*, ***kwargs*)

Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

tex_fmt(*content=None*, *attributes=None*, *mathmode=False*, *level=1*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

exception disseminate.tags.caption.**CaptionError**
Bases: `Exception`

An error was encountered in processing a caption.

13.12.4 Code

Code formatting tags

class disseminate.tags.code.**Code**(**args*, ***kwargs*)
Bases: *disseminate.tags.tag.Tag*

A tag for displaying code.

html_fmt(*content=None*, *attributes=None*, *format_func='html_fmt'*, *method='html'*, *level=1*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

property lexer
Get the lexer for code highlighting.

tex_fmt(*content=None*, *attributes=None*, *mathmode=False*, *level=1*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

class disseminate.tags.code.**Dm**(**args*, ***kwargs*)
Bases: *disseminate.tags.code.Code*

A tag for displaying dm code

class disseminate.tags.code.**Html**(**args*, ***kwargs*)
Bases: *disseminate.tags.code.Code*

A tag for displaying html code

class disseminate.tags.code.**Java**(**args*, ***kwargs*)
Bases: *disseminate.tags.code.Code*

A tag for displaying ruby code

class disseminate.tags.code.**Javascript**(**args*, ***kwargs*)
Bases: *disseminate.tags.code.Code*

A tag for displaying javascript code

class disseminate.tags.code.**Python**(**args*, ***kwargs*)
Bases: *disseminate.tags.code.Code*

A tag for displaying python code

class disseminate.tags.code.**Ruby**(**args*, ***kwargs*)
Bases: *disseminate.tags.code.Code*

A tag for displaying ruby code

13.12.5 Data

Tags for data sources

class disseminate.tags.data.**Cell**(*name*, *content*, *attributes*, *context*)

Bases: *disseminate.tags.tag.Tag*

A cell in a table

tex_fmt(***kwargs*)

Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

class disseminate.tags.data.**Data**(*name*, *content*, *attributes*, *context*)

Bases: *disseminate.tags.tag.Tag*

A container for data.

Attributes

data [Dict[Union[str, int], Tuple(str)]] A data dict with the column names (str) or column index (int) as keys and the rows (tuple) as values.

processed_headers [Union[List[`tags.Tag`], None]] The headers processed into tags

processed_rows [List[List[`tags.tag`]]] The rows processed into tags

property headers

The tuple of the data headers

abstract load(*filepath_or_buffer*)

Load the dataframe from a filepath or a string buffer.

Parameters

filepath_or_buffer [Union[`pathlib.Path`, str, `io.StringIO`]] The filename and path (filepath) or string buffer.

property num_cols

The number of columns in the data

property num_rows

The number of columns in the data

process_tags()

Process the tags for the items in the tag's data.

property rows

An iterator for the data rows

property rows_transpose
An iterator for the data rows (transposed)

class disseminate.tags.data.DelimData(*delimiter=','*, **args*, ***kwargs*)
Bases: disseminate.tags.data.Data

A container for delimiter data (ex: comma-separated values)

load(*filepath_or_buffer*, *delimiter=None*)
Load the dataframe from a filepath or a string buffer.

Parameters

filepath_or_buffer [Union[pathlib.Path, str, io.StringIO]] The filename and path (filepath) or string buffer.

class disseminate.tags.data.HeaderCell(*name*, *content*, *attributes*, *context*)
Bases: disseminate.tags.tag.Tag

A header cell in a table

tex_fmt(***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, Tag]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, Attributes]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

13.12.6 Eq

Tags to render equations

class disseminate.tags.eqs.Eq(**args*, *block_equation=False*, ***kwargs*)
Bases: disseminate.tags.img.Img

The inline equation tag

Render an equation in native LaTeX (.tex targets) or into a rendered SVG image using LaTeX (.html targets).

Attributes

aliases [Tuple[str]] A list of strs for other names a tag goes by

active [bool] If True, the Tag can be used by the TagFactory.

block_equation [bool] If True, the equation will be rendered as a LaTeX block equation using a math environment. ex: begin{equation}...end{equation} If False, the equation will be rendered as a LaTeX inline equation. ex: "ensuremath{y=x}

html_fmt(*attributes=None*, *context=None*, *method='html'*, ***kwargs*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

tex_fmt(*content=None*, *attributes=None*, *mathmode=False*, ***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

13.12.7 FeatureBox

A tag for a feature box used in some textbooks (like examples)

class disseminate.tags.featurebox.**ExampleBox**(*name*, **args*, ***kwargs*)
Bases: *disseminate.tags.featurebox.FeatureBox*

A box for examples

class disseminate.tags.featurebox.**FeatureBox**(*name*, **args*, ***kwargs*)
Bases: *disseminate.tags.tag.Tag*

A box with a feature for some texts, like an Example or Note box.

Attributes

active [bool] This tag is active.

include_paragraphs [bool] The contents of this tag cannot be included in paragraphs.

html_classes [str] Classes to append in the html tag.

html_fmt(*attributes=None*, ***kwargs*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

`class disseminate.tags.featurebox.`**`ProblemBox`**(*name*, **args*, ***kwargs*)

Bases: *disseminate.tags.featurebox.FeatureBox*

A box for problems

13.12.8 Figs

Tags for figure environments.

`class disseminate.tags.figs.`**`BaseFigure`**(**args*, ***kwargs*)

Bases: *disseminate.tags.tag.Tag*

A base class for a figure tag.

The BaseFigure initializes figures by adding 'id' attributes to labels in the label manager and reorganizing captions to the bottom of the figure.

`html_fmt`(*attributes=None*, *method='html'*, ***kwargs*)

Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

`class disseminate.tags.figs.`**`Figure`**(**args*, ***kwargs*)

Bases: *disseminate.tags.figs.BaseFigure*

The @figure/@fig tag

class disseminate.tags.figs.**FullFigure**(**args*, ***kwargs*)
Bases: *disseminate.tags.figs.BaseFigure*

The @fullfigure/@ffig tag

class disseminate.tags.figs.**MarginFigure**(**args*, ***kwargs*)
Bases: *disseminate.tags.figs.BaseFigure*

The @marginfig tag

xhtml_fmt(*attributes=None*, ***kwargs*)
Convert the tag to an xhtml string or html element.

Returns

xhtml [str or xhtml element] A string in XHTML format or an XHTML element (`lxml.builder.E`).

class disseminate.tags.figs.**Panel**(*name*, *content*, *attributes*, *context*)
Bases: *disseminate.tags.tag.Tag*

A panel (sub-figure) for a figure.

html_fmt(*attributes=None*, *method='html'*, ***kwargs*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

tex_fmt(*attributes=None*, ***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

13.12.9 Headings

Tags for headings.

class disseminate.tags.headings.**Chapter**(*name*, *content*, *attributes*, *context*, ***kwargs*)
Bases: *disseminate.tags.headings.Heading*

class disseminate.tags.headings.**Heading**(*name*, *content*, *attributes*, *context*, ***kwargs*)
Bases: *disseminate.tags.tag.Tag*, *disseminate.tags.label.LabelMixin*

A heading tag.

Note: If the content isn't specified and an entry exists in the context with the tag's name, then this tag's content will be replaced with the contents from the context.

---

default_fmt(*content=None*, *attributes=None*)
Convert the tag to a text string.

Strips the tag information and simply return the content of the tag.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

Returns

text_string [str] A text string with the tags stripped.

generate_label_id()
Generate the label_id to use in creating the label.

Override this function to customize the generation of the label_id.

Returns

label_id [str] The generated label_id.

generate_label_kind()
Generate the kind tuple for the created label.

Override this function to customize the generation of the label kind.

Returns

kind [Tuple[str]] A tuple of strings for the kind of label. ex: ('heading', 'chapter')

html_fmt(*content=None*, ***kwargs*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

tex_fmt(*content=None*, *attributes=None*, *mathmode=False*, *level=1*, ***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

class disseminate.tags.headings.**Para**(*name*, *content*, *attributes*, *context*)
Bases: *disseminate.tags.tag.Tag*

A paragraph heading tag.

class disseminate.tags.headings.**Part**(*name*, *content*, *attributes*, *context*, ***kwargs*)
Bases: *disseminate.tags.headings.Heading*

class disseminate.tags.headings.**Section**(*name*, *content*, *attributes*, *context*, ***kwargs*)
Bases: *disseminate.tags.headings.Heading*

A section heading tag.

class disseminate.tags.headings.**SubSection**(*name*, *content*, *attributes*, *context*, ***kwargs*)
Bases: *disseminate.tags.headings.Heading*

A subsection heading tag.

class disseminate.tags.headings.**SubSubSection**(*name*, *content*, *attributes*, *context*, ***kwargs*)
Bases: *disseminate.tags.headings.Heading*

A subsubsection heading tag.

class disseminate.tags.headings.**Title**(*name*, *content*, *attributes*, *context*, ***kwargs*)
Bases: *disseminate.tags.headings.Heading*

13.12.10 Img

Image tags

class disseminate.tags.img.**Img**(**args*, ***kwargs*)

Bases: *disseminate.tags.tag.Tag*

The img tag for inserting images.

When rendering to a target document format, this tag may use a converter, the attributes and context of this tag to convert the infile to an outfile in a needed format.

Attributes

active [bool] If True, the Tag can be used by the TagFactory.

html_name [str] If specified, use this name when rendering the tag to html. Otherwise, use name.

in_ext [Optional[str]] Used to correctly identify the builder to add if the contents need to be rendered first.

img_filepath [str] The path for the (source) image.

add_file(*target*, *content=None*, *attributes=None*, *context=None*)

Convert and add the file dependency for the specified document target.

Parameters

target [str] The document target format. ex: '.html', '.tex'

content [Optional[Union[str, List[Union[str, list, *Tag*], *Tag*]]] The contents of the tag. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] The attributes of the tag.

context [Optional[*Type[BaseContext]*]] The context with values for the document.

Returns

outfilepath [*paths.TargetPath*] The filepath for the file in the target document directory.

Raises

ImgFileNotFound Raises an ImgFileNotFound exception if a filepath couldn't be found in the tag contents.

BuildError If a builder could not be found for the builder

content_as_filepath(*content=None*, *context=None*)

Returns a filepath from the content, if it's a valid filepath, or returns None if it isn't.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*], *Tag*]]] The contents of the tag. It can either be a string, a tag or a list of strings, tags and lists.

context [Optional[*Type[BaseContext]*]] The context with values for the document.

Returns

filepath [Union[pathlib.Path. None]] The filepath, if found, or None if a valid filepath was not found.

html_fmt(*content=None*, *attributes=None*, *context=None*, *method='html'*, ***kwargs*)

Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

tex_fmt(*content=None*, *attributes=None*, *context=None*, ***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

exception disseminate.tags.img.**ImgFileNotFound**
Bases: *disseminate.tags.exceptions.TagError*

The image file was not found.

13.12.11 Label

The Label tag to reference captions and other labels.

class disseminate.tags.label.**LabelAnchor**(**args*, *content*, ***kwargs*)
Bases: *disseminate.tags.tag.Tag*

The label anchor.

html_fmt(*method='html'*, *level=1*, ***kwargs*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

class disseminate.tags.label.**LabelMixin**(**args*, ***kwargs*)

Bases: `object`

A mixin class for working with LabelTags LabelAnchors.

This mixing works in coordination with the *Tag* base class. The constructor (__init__) of this class should be run after the Tag base class's constructor.

create_label()

Create the label in the label_manager for this tag.

generate_label_id()

Generate the label_id to use in creating the label.

Override this function to customize the generation of the label_id.

Returns

label_id [str] The generated label_id.

generate_label_kind()

Generate the kind tuple for the created label.

Override this function to customize the generation of the label kind.

Returns

kind [Tuple[str]] A tuple of strings for the kind of label. ex: ('heading', 'chapter')

class disseminate.tags.label.**LabelTag**(*name*, *attributes*, *content*, *context*, ***kwargs*)

Bases: *disseminate.tags.tag.Tag*

The name portion of a label.

ex: Chap. 1.2 or Fig. 1.

Parameters

content [str] The label_id of an existing label.

kind [Tuple[str]] The kind of the label is a tuple that identified the kind of a label from least specific to most specific. ex: ('figure',), ('chapter',), ('equation',), ('heading', 'h1',) See `LabelManager.format_string`

prepend_id [Optional[str]] If specified, the label id (identifier) will have this string prepended to it.

(see the :class:`Tag <disseminate.tags.tag.Tag>` base class for details on the other parameters)

default_fmt(*content=None*, *attributes=None*)

Convert the tag to a text string.

Strips the tag information and simply return the content of the tag.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

Returns

text_string [str] A text string with the tags stripped.

html_fmt(*content=None*, *attributes=None*, *format_func='html_fmt'*, *method='html'*, *level=1*, ***kwargs*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

process_content = False
Do not convert typography characters. Conversion of typography characters might cause issues with matching the label id.

tex_fmt(*content=None*, *attributes=None*, *mathmode=False*, *level=1*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

xhtml_fmt(*format_func='xhtml_fmt'*, *method='xhtml'*, ***kwargs*)
Convert the tag to an xhtml string or html element.

Returns

xhtml [str or xhtml element] A string in XHTML format or an XHTML element (`lxml.builder.E`).

disseminate.tags.label.**create_label**(*tag*, *kind*, *label_id=None*)
Create a label in the label_manager for the given tag.

Note: This function does not create a *LabelTag*. Only a `Label` object through the document's :obj:<LabelManager <disseminate.labels.label_manager.LabelManager>`

Parameters

tag [*Tag*] The tag creating the label.

kind [Tuple[str]] The kind of label. ex: ('heading', 'section')

label_id [Optional[str]] The identifier for the label to create. If None is specified, the generate_label_id function will be used.

Returns

label_id [Union[str, None]] The label_id of the created label, if successful. None if a label could not be created.

disseminate.tags.label.**generate_label_id**(*tag*)
Generate a label_id string for the given tag.

Parameters

tag [*Tag*] The tag creating the label.

Returns

label_id [str] The label id string for the tag.

13.12.12 List

Tags for lists.

class disseminate.tags.list.**List**(*name*, *content*, *attributes*, *context*, ***kwargs*)
Bases: *disseminate.tags.tag.Tag*

A tag for lists

html_fmt(*content=None*, *attributes=None*, *format_func='html_fmt'*, *method='html'*, *level=1*, ***kwargs*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

tex_fmt(*attributes=None*, ***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

class disseminate.tags.list.**ListItem**(**args*, ***kwargs*)
Bases: *disseminate.tags.tag.Tag*

An item in a list

tex_fmt(***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

class disseminate.tags.list.**OrderedList**(*name*, *content*, *attributes*, *context*, ***kwargs*)
Bases: *disseminate.tags.list.List*

A tag for ordered lists

disseminate.tags.list.**clean_string_list**(*parsed_list*)
Clean the string list created by parse_string_list.

Cleaning include removeing extra spaces and newlines in parse string line elements.

Parameters

parsed_list [List[Tuple[int, str]]] The parsed list from parse_string_list.

Returns

cleaned_list [List[Tuple[int, str]]] The cleaned list.

```
>>> l = parse_string_list("- This is my first item.\n"
    ..
```

... " - This is my first subitemn")

```
>>> clean_string_list(l)
    ..
```

[(0, 'This is my first item.'), (2, 'This is my first subitem')]

disseminate.tags.list.**normalize_levels**(*parsed_list*, *list_level_spaces=2*)

Normalize the levels from a parse_list so that the first level is 0, and subsequent levels are properly incremented.

Parameters

parsed_list [List[Tuple[int, str]]] The parsed list from parse_string_list.

list_level_spaces [int] The number of spaces used to identify sub-levels in a list.

Returns

normalized_list [List[Tuple[int, str]]] The normalized list with levels fixed.

```
>>> l = parse_string_list("- This is my first item.\n"
    ..
```

... " - This is my first subitemn")

```
>>> normalize_levels(l)
    ..
```

[(0, 'This is my first item.'), (1, 'This is my first subitem')]

disseminate.tags.list.**parse_list**(*content*, *context*)

Parse lists (and sublists) from a string or list of content.

Parameters

content [Union[str, List[Union[str, list, *Tag*], *Tag*]] The contents of the tag. It can either be a string, a tag or a list of strings, tags and lists.

context [`document.DocumentContext`] The document's context dict.

Returns

returned_list [list] The list of parse list items, to be used for the content of a list.

disseminate.tags.list.**parse_string_list**(*s*)

Parse a string with lists.

Parameters

s [str] The string to parse

Returns

parsed_list [List[Tuple[int, str]]]

Examples

```
>>> parse_string_list("- This is my first item.\n"
...                   "  - This is my first subitem\n")
[(0, 'This is my first item.'), (2, 'This is my first subitem')]
```

13.12.13 Navigation

Navigation tags

class disseminate.tags.navigation.**Epublink**(*name*, *content*, ***kwargs*)
Bases: *disseminate.tags.navigation.OtherLink*

Produces a link to the epub target for the root document.

other_filepath(*target=None*)
The filepath for the other document target or src_filepath to create a link to.

class disseminate.tags.navigation.**Next**(*name*, *content*, ***kwargs*)
Bases: *disseminate.tags.ref.Ref*

Produces links to the next document.

Notes

1. As opposed to the parent Ref tag, if a label is not found, an empty string is returned by the target format functions.
2. The following are tag attributes that are recognized:

kind [str] The kind of label to reference by this tag. By default, it's a heading.

default_fmt(***kwargs*)
Convert the tag to a text string.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

cache [Optional[dict]] If specified, the cache values will be used instead of being evaluated. Possibilities: - 'label': `types.Label` - 'documents_by_id': Dict[str, `document.Document`]

Returns

text_string [str] A text string with the tags stripped.

html_fmt(*cache=None*, ***kwargs*)
Convert the tag to an (x)html string or (x)html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

cache [Optional[dict]] If specified, the cache values will be used instead of being evaluated. Possibilities: - 'label': `types.Label` - 'documents_by_id': Dict[str, `document.Document`] - 'format_str': str

format_func [Optional[str]] The tag format function to use in rendering the reference tag. (ex: 'html_fmt' or 'xhtml_fmt')

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

tex_fmt(*cache=None*, ***kwargs*)

Format the tag in LaTeX format.

Note: This function renders tex links to compiled pdf documents

---

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

cache [Optional[dict]] If specified, the cache values will be used instead of being evaluated. Possibilities: - 'label': `types.Label` - 'documents_by_id': Dict[str, `document.Document`]

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

xhtml_fmt(***kwargs*)

Convert the tag to an xhtml string or html element.

Returns

xhtml [str or xhtml element] A string in XHTML format or an XHTML element (`lxml.builder.E`).

class disseminate.tags.navigation.**OtherLink**(*name*, *content*, ***kwargs*)

Bases: *disseminate.tags.tag.Tag*

Produces a link to a source document or another target for this document.

default_fmt(***kwargs*)

Convert the tag to a text string.

Strips the tag information and simply return the content of the tag.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

Returns

text_string [str] A text string with the tags stripped.

html_fmt(***kwargs*)

Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

other_filepath(*target=None*)

The filepath for the other document target or src_filepath to create a link to.

other_relpath(*this_target*, *other_target=None*)

Produce the the relpath for the other target.

tex_fmt(***kwargs*)

Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

this_filepath(*target*)

The filepath for this (target) document.

xhtml_fmt(***kwargs*)

Convert the tag to an xhtml string or html element.

Returns

xhtml [str or xhtml element] A string in XHTML format or an XHTML element (`lxml.builder.E`).

class disseminate.tags.navigation.**Pdflink**(*name*, *content*, ***kwargs*)
Bases: *disseminate.tags.navigation.OtherLink*

Produces a link to the pdf target for this document

class disseminate.tags.navigation.**Prev**(*name*, *content*, ***kwargs*)
Bases: *disseminate.tags.ref.Ref*

Produces links to the previous document.

Notes

1. As opposed to the parent Ref tag, if a label is not found, an empty string is returned by the target format functions.
2. The following are tag attributes that are recognized

kind [str] The kind of label to reference by this tag. By default, it's a heading.

default_fmt(***kwargs*)
Convert the tag to a text string.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

cache [Optional[dict]] If specified, the cache values will be used instead of being evaluated. Possibilities: - 'label': `types.Label` - 'documents_by_id': Dict[str, `document.Document`]

Returns

text_string [str] A text string with the tags stripped.

html_fmt(*cache=None*, ***kwargs*)
Convert the tag to an (x)html string or (x)html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

cache [Optional[dict]] If specified, the cache values will be used instead of being evaluated. Possibilities: - 'label': `types.Label` - 'documents_by_id': Dict[str, `document.Document`] - 'format_str': str

format_func [Optional[str]] The tag format function to use in rendering the reference tag. (ex: 'html_fmt' or 'xhtml_fmt')

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

tex_fmt(*cache=None*, ***kwargs*)
Format the tag in LaTeX format.

Note: This function renders tex links to compiled pdf documents

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

cache [Optional[dict]] If specified, the cache values will be used instead of being evaluated. Possibilities: - 'label': `types.Label` - 'documents_by_id': Dict[str, `document.Document`]

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

xhtml_fmt(***kwargs*)
Convert the tag to an xhtml string or html element.

Returns

xhtml [str or xhtml element] A string in XHTML format or an XHTML element (`lxml.builder.E`).

class disseminate.tags.navigation.**Srclink**(*name*, *content*, ***kwargs*)
Bases: *disseminate.tags.navigation.OtherLink*

Produces a link to the src file for this document

other_filepath(*target=None*)
The filepath for the other document target or src_filepath to create a link to.

class disseminate.tags.navigation.**Texlink**(*name*, *content*, ***kwargs*)
Bases: *disseminate.tags.navigation.OtherLink*

Produces a link to the tex target for this document.

class disseminate.tags.navigation.**Txtlink**(*name*, *content*, ***kwargs*)
Bases: *disseminate.tags.navigation.OtherLink*

Produces a link to the txt target for this document.

Receivers

Navigation tags

disseminate.tags.navigation.**set_navigation_labels**(*root_document*)
Set the navigation labels in the context of all documents in a document tree.

13.12.14 Notes

Tags for note environments.

class disseminate.tags.notes.**Sidenote**(*name*, *content*, *attributes*, *context*)
Bases: *disseminate.tags.tag.Tag*

A sidenote tag.

13.12.15 Preamble

Tags for document preambles.

class disseminate.tags.preamble.**Authors**(*name*, *context*, ***kwargs*)
Bases: *disseminate.tags.tag.Tag*

A tag for listing the author or authors.

author_string()
Generate a formatted string listing the authors.

html_fmt(*content=None*, *method='html'*, *level=1*, **args*, ***kwargs*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

tex_fmt(**args*, ***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

class disseminate.tags.preamble.**Titlepage**(*name*, *content*, *attributes*, *context*)

Bases: *disseminate.tags.tag.Tag*

A titlepage tag.

property **authors**

The authors of the document

html_fmt(*content=None*, *attributes=None*, *format_func='html_fmt'*, *method='html'*, *level=1*, ***kwargs*)

Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

tex_fmt(*content=None*, *attributes=None*, *mathmode=False*, *level=1*, ***kwargs*)

Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

property **title**

The title of the project

13.12.16 Ref

The Ref tag to reference captions and other labels.

class disseminate.tags.ref.**Ref**(*name*, *content*, **args*, ***kwargs*)

Bases: *disseminate.tags.tag.Tag*

A tag to reference a label.

Attributes

doc_id [str] The doc_id for the document that owns the label referenced. This might be different to the doc_id listed in the context if a reference is made to a label in another document.

label_id [str] The id of the label referenced by this tag.

.. note:: The ref tag takes the label id as the content. If the label is not found, a label manager exception is raised. (*label_manager.exceptions.LabelNotFound*)

default_fmt(*content=None*, *attributes=None*, *cache=None*, ***kwargs*)

Convert the tag to a text string.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

cache [Optional[dict]] If specified, the cache values will be used instead of being evaluated. Possibilities: - 'label': `types.Label` - 'documents_by_id': Dict[str, `document.Document`]

Returns

text_string [str] A text string with the tags stripped.

document(*cache=None*)

The document that owns the label referenced by this tag.

Parameters

cache [Optional[dict]] If specified, the cache values will be used instead of being evaluated. Possibilities: - 'label': `types.Label` - 'documents_by_id': Dict[str, `document.Document`]

Returns

document [Union[*Document*, None]] The document that owns the label referenced by this tag, if available. None, if a document could not be found.

html_fmt(*content=None*, *attributes=None*, *cache=None*, *format_func='html_fmt'*, *method='html'*, *level=1*, ***kwargs*)

Convert the tag to an (x)html string or (x)html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

cache [Optional[dict]] If specified, the cache values will be used instead of being evaluated. Possibilities: - 'label': `types.Label` - 'documents_by_id': Dict[str, `document.Document`] - 'format_str': str

format_func [Optional[str]] The tag format function to use in rendering the reference tag. (ex: 'html_fmt' or 'xhtml_fmt')

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

`property label`
Retrieve the label for this ref tag.

`tex_fmt`(*content=None*, *attributes=None*, *mathmode=False*, *cache=None*, *level=1*, ***kwargs*)
Format the tag in LaTeX format.

Note: This function renders tex links to compiled pdf documents

---

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

cache [Optional[dict]] If specified, the cache values will be used instead of being evaluated. Possibilities: - 'label': `types.Label` - 'documents_by_id': Dict[str, `document.Document`]

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

`url`(*target='.html'*, *include_anchor=True*, *cache=None*)
The url path for the document referenced by the label for this tag.

Parameters

target [Optional[str]] The target extension for the target file. ex: '.html' or '.tex'

include_anchor [Optional[bool]] If True (default), the html link anchor will be appended to the url path.

cache [Optional[dict]] If specified, the cache values will be used instead of being evaluated. Possibilities: - 'label': `types.Label` - 'documents_by_id': Dict[str, `document.Document`]

Returns

url_path [str] The url path for the document owning the label referenced by this tag.

exception disseminate.tags.ref.**RefError**
Bases: Exception

A reference to a document could not be found.

Receivers

The Ref tag to reference captions and other labels.

disseminate.tags.ref.**add_ref_labels**(*builder*, ***kwargs*)
Find and add the labels associated with Ref tags for all tags in the context.

13.12.17 Table

Tags for tables

class disseminate.tags.table.**BaseTable**(**args*, ***kwargs*)
Bases: *disseminate.tags.tag.Tag*

A base tag for all tables

html_fmt(*content=None*, *attributes=None*, *format_func='html_fmt'*, *method='html'*, *level=1*, ***kwargs*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

tex_fmt(*content=None*, *attributes=None*, *mathmode=False*, *level=1*, ***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

xhtml_fmt(*format_func='xhtml_fmt'*, *method='xhtml'*, ***kwargs*)
Convert the tag to an xhtml string or html element.

Returns

xhtml [str or xhtml element] A string in XHTML format or an XHTML element (`lxml.builder.E`).

class disseminate.tags.table.**FullTable**(**args*, ***kwargs*)
Bases: *disseminate.tags.table.BaseTable*

The @fulltable tag

class disseminate.tags.table.**MarginTable**(**args*, ***kwargs*)
Bases: *disseminate.tags.table.BaseTable*

The @margintable tag

class disseminate.tags.table.**Table**(**args*, ***kwargs*)
Bases: *disseminate.tags.table.BaseTable*

The @table tag

13.12.18 Text

Text formatting tags

class disseminate.tags.text.**Body**(*name*, *content*, *attributes*, *context*)
Bases: *disseminate.tags.tag.Tag*

A body tag for the body of a document.

In (x)html, this is rendered as a instead of the default

html_fmt(*attributes=None*, ***kwargs*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

xhtml_fmt(*content=None*, *attributes=None*, *format_func='xhtml_fmt'*, *method='xhtml'*, *level=1*, ***kwargs*)
Convert the tag to an xhtml string or html element.

Returns

xhtml [str or xhtml element] A string in XHTML format or an XHTML element (`lxml.builder.E`).

class disseminate.tags.text.**Bold**(*name*, *content*, *attributes*, *context*)
Bases: *disseminate.tags.tag.Tag*

A bold tag.

Attributes

aliases [Tuple[str]] A list of strs for other names a tag goes by

html_name [str] If specified, use this name when rendering the tag to html. Otherwise, use name.

tex_cmd [str] Use this name to render the tex command.

active [bool] This tag is active.

class disseminate.tags.text.**Italics**(*name*, *content*, *attributes*, *context*)
Bases: *disseminate.tags.tag.Tag*

An italics tag.

Attributes

aliases [Tuple[str]] A list of strs for other names a tag goes by

html_name [str] If specified, use this name when rendering the tag to html. Otherwise, use name.

tex_cmd [str] Use this name to render the tex command.

active [bool] This tag is active.

class disseminate.tags.text.**P**(*name*, *content*, *attributes*, *context*)
Bases: *disseminate.tags.tag.Tag*

A Paragraph tag

Attributes

active [bool] This tag is active.

include_paragraphs [bool] The contents of this tag can be included in paragraphs.

tex_fmt(***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

class disseminate.tags.text.**Sub**(*name*, *content*, *attributes*, *context*)
Bases: *disseminate.tags.tag.Tag*

A subscript tag.

Attributes

html_name [str] If specified, use this name when rendering the tag to html. Otherwise, use name.

active [bool] This tag is active.

tex_fmt(*content=None*, *attributes=None*, *mathmode=False*, *level=1*, ***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

class disseminate.tags.text.**Sup**(*name*, *content*, *attributes*, *context*)
Bases: *disseminate.tags.tag.Tag*

A superscript tag.

Attributes

html_name [str] If specified, use this name when rendering the tag to html. Otherwise, use name.

active [bool] This tag is active.

tex_fmt(*content=None*, *attributes=None*, *mathmode=False*, *level=1*, ***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

class disseminate.tags.text.**Supsub**(**args*, ***kwargs*)
Bases: *disseminate.tags.tag.Tag*

A superscript/subscript tag, together, that displays them one on top of the other.

The content of the tag consists of two elements separated by a '&&' character. ex: @supsub{superscript && subscript}

Attributes

active [bool] This tag is active.

html_fmt(*content=None*, *attributes=None*, *format_func='html_fmt'*, *method='html'*, *level=1*, ***kwargs*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

tex_fmt(*content=None*, *attributes=None*, *mathmode=False*, *level=1*, ***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

class disseminate.tags.text.**Symbol**(*name*, *content*, *attributes*, *context*)
Bases: *disseminate.tags.tag.Tag*

One or more greek characters.

Attributes

aliases [Tuple[str]] A list of strs for other names a tag goes by

active [bool, default: True] This tag is active.

html_fmt(*content=None*, *attributes=None*, *method='html'*, *level=1*, ***kwargs*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

tex_fmt(*content=None*, *attributes=None*, *mathmode=False*, *level=1*, ***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

xhtml_fmt(*content=None*, *attributes=None*, *method='xml'*, *level=1*, ***kwargs*)
Convert the tag to an xhtml string or html element.

Returns

xhtml [str or xhtml element] A string in XHTML format or an XHTML element (`lxml.builder.E`).

class disseminate.tags.text.**Verb**(*name*, *content*, *attributes*, *context*)
Bases: *disseminate.tags.tag.Tag*

A verbatim tag for displaying unformatted blocks of text.

Attributes

aliases [Tuple[str]] A list of strs for other names a tag goes by

html_name [str] If specified, use this name when rendering the tag to html. Otherwise, use name.

active [bool] This tag is active.

process_content [bool] Do not process the contents of the tag–just take the contents literally.

include_paragraphs [bool] The contents of this tag cannot be included in paragraphs.

html_fmt(*attributes=None*, **args*, ***kwargs*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

`tex_fmt`(**args*, ***kwargs*)

Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

13.12.19 TOC

Formatting of Table of Contents for documents

`class disseminate.tags.toc.Toc`(*name*, *content*, *attributes*, *context*)

Bases: *disseminate.tags.tag.Tag*

Table of contents and listings.

contents [str] The contents are the label types to list. The following entries are supported:

- document: list the document labels
- heading : list heading labels
- figure : list figure labels
- table : list table labels

Additionally, the following modifiers have special meaning: - all : list labels from all documents - current : list labels from the current document (default)

- **expanded** [pertains to 'all document' and 'all headings'.] Show all label references for all documents.
- **abbreviated** [pertains to 'all document' and 'all headings'.] show all label references for the current document and the top level heading for other documents.

- **collapsed** [pertains to 'all document' and 'all headings'.] show only the documents without headings. (default)

`get_labels`()
Get the labels, ordering function and labeling type.

Returns

labels [List[`label_manager.types.Label`]] The labels referenced by this TOC.

`html_fmt`(*content=None*, *attributes=None*, *cache=None*, *format_func='html_fmt'*, *method='html'*, *level=1*, ***kwargs*)
Convert the tag to an html string or html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

format_func [Optional[str]] The format function to use with by formatted_content when formatting sub-tags.

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

`property reference_tags`
This tag's TocRef tag items.

`tex_fmt`(*content=None*, *attributes=None*, *mathmode=False*, *level=1*, ***kwargs*)
Format the tag in LaTeX format.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

`exception` disseminate.tags.toc.**TocError**
Bases: `Exception`

An error was encountered while processing a table of contents tag.

`class` disseminate.tags.toc.**TocRef**(*name*, *content*, **args*, ***kwargs*)
Bases: *disseminate.tags.ref.Ref*

A Ref tag for the TOC.

This is a separate class so that the label_fmt may be different for TOC entries.

html_fmt(*content=None*, *attributes=None*, *cache=None*, *format_func='html_fmt'*, *method='html'*, *level=1*, ***kwargs*)

Convert the tag to an (x)html string or (x)html element.

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

cache [Optional[dict]] If specified, the cache values will be used instead of being evaluated. Possibilities: - 'label': `types.Label` - 'documents_by_id': Dict[str, `document.Document`] - 'format_str': str

format_func [Optional[str]] The tag format function to use in rendering the reference tag. (ex: 'html_fmt' or 'xhtml_fmt')

method [Optional[str]] The rendering method for the string. ex: 'html' or 'xml'

level [Optional[int]] The level of the tag.

Returns

html [str or html element] A string in HTML format or an HTML element (`lxml.builder.E`).

tex_fmt(*content=None*, *attributes=None*, *mathmode=False*, *cache=None*, *level=1*, ***kwargs*)

Format the tag in LaTeX format.

Note: This function renders tex links to compiled pdf documents

Parameters

content [Optional[Union[str, List[Union[str, list, *Tag*]]]] Specify an alternative content from the tag's content. It can either be a string, a tag or a list of strings, tags and lists.

attributes [Optional[Union[str, *Attributes*]]] Specify an alternative attributes dict from the tag's attributes. It can either be a string or an attributes dict.

mathmode [Optional[bool]] If True, the tag will be rendered in math mode. Otherwise (default) latex text mode is assumed.

cache [Optional[dict]] If specified, the cache values will be used instead of being evaluated. Possibilities: - 'label': `types.Label` - 'documents_by_id': Dict[str, `document.Document`]

level [Optional[int]] The level of the tag.

Returns

tex_string [str] The formatted tex string.

13.12.20 Exceptions

Exceptions for Tag objects

exception disseminate.tags.exceptions.**TagError**
Bases: Exception

An error was encountered while interpreting a tag.

13.12.21 Signals

Tag event signals

disseminate.tags.signals.**tag_created = <disseminate.signals.signals.Signal object>**
A signal emitted when a tag is created. Receivers take a tag parameter.

13.12.22 Receivers

Signal: tag_created

The receivers for tag events.

disseminate.tags.receivers.**process_content**(*tag*, *tag_factory*, ***kwargs*)
A receiver to parse the contents of tags into sub-tags.

disseminate.tags.receivers.**process_macros**(*tag*, ***kwargs*)
A receiver for replacing macros in pre-parsed tag strings.

disseminate.tags.receivers.**process_paragraphs**(*tag*, *tag_factory*, *tag_base_cls*, ***kwargs*)
A receiver to parse the paragraphs of tags.

disseminate.tags.receivers.**process_typography**(*tag*, *tag_base_cls*, ***kwargs*)
A receiver to parse the typography of tags.

13.12.23 Utils

Misc utilities for tags.

disseminate.tags.utils.**content_to_str**(*content*, *target='.txt'*, ***kwargs*)
Convert a tag or string to a string for the specified target.

This function is used to convert an element, which is either a string, tag or list of strings and tags, to a string.

Parameters

content [Union[str, List[Union[str, list, *Tag*], *Tag*]] The element to convert into a string.

target [str, optional] The target format of the string to return.

Returns

formatted_str [str] A string in the specified target format

disseminate.tags.utils.**copy_tag**(*tag*)
Create a copy of the given tag.

The tag, attributes and content are deep copies, and the context points to the same context as the given tag.

Parameters

tag [Union[str, list Tag]] The tag to copy.

Returns

tag_copy [Tag] The tag copy.

disseminate.tags.utils.**format_content**(*content*, *format_func*, ***kwargs*)
Format the content using the format_func.

Parameters

content [Union[str, List[Union[str, list, Tag]], optional] The content to format

format_func [str] The name of the format_func to use. ex: 'tex_fmt'

Returns

content [str or list] The content formatted using the specified format_func.

disseminate.tags.utils.**percentage**(*value*)
Given a strings (or floats or ints), convert into a float number between [0.0, 100.0].

Parameters

value [Union[str, int, float]]

Returns

percentage [Union[int, None]] The percentage from 0 to 100, or None if the value could not be converted.

Examples

```
>>> percentage('10.2%')
11
>>> percentage('10.2')
1020
>>> percentage('0.3%')
1
>>> percentage('0.3')
30
>>> percentage(30)
30
```

disseminate.tags.utils.**repl_tags**(*element*, *tag_class*, *replacement*)
Replace all instances of a tag class with a replacement string.

Parameters

element [Union[str, list Tag]] The element to replace tags with a replacement string.

tag_class [Tag] A tag class or subclass to replace.

replacement [str] The string to replace the tag with.

disseminate.tags.utils.**replace_context**(*tag*, *new_context*)
Replace the context for the given tag (and all subtags) to the given new_context.

Parameters

tag [Tag] The tag to replace the context.

new_context [*Type[BaseContext]*] The new context to replace.

disseminate.tags.utils.**tex_percentwidth**(*attributes*, *target='.tex'*, *use_positional=False*)

Generates an tex width string based on the 'width' entry in the attributes.

Currently, this function works in halves (w50), thirds (w33, w66) and quarters (w25, w75)

Parameters

attributes [*attributes.Attributes*] The attributes dict

target [Optional[str]] The target format to format the attribute dict for.

use_positional [Optional[bool]] Insert the entry as a positional attribute

Returns

attributes: *attributes.Attributes* The same attributes dict with the width inserted as a StringPositionalArgument, if a width was found.

Examples

```
>>> tex_percentwidth('width=50%')
Attributes{'width': '0.49\\textwidth'}
>>> tex_percentwidth('width=3.2in')
Attributes{'width': '3.2in'}
>>> tex_percentwidth('width=50%', use_positional=True)
Attributes{'width': '50%', '0.49\\textwidth': <class '...StringPositionalValue'>}
```

disseminate.tags.utils.**xhtml_percentwidth**(*attributes*, *target='.html'*)

Generates an (x)html class name based on the 'width' entry in the attributes.

Currently, this function works in halves (w50), thirds (w33, w66) and quarters (w25, w75)

Parameters

attributes [*attributes.Attributes*] The attributes dict

target [Optional[str]] The target format to format the attribute dict for.

Returns

attributes: *attributes.Attributes* The same attributes dict with the 'class' entry set to an html class for the percent width, if a width was found.

Examples

```
>>> xhtml_percentwidth('width=50%')
Attributes{'width': '50%', 'class': 'w50'}
>>> xhtml_percentwidth('width=0%')  # None returned
Attributes{'width': '0%'}
```

13.13 Utilities

Extra utilities for Python

13.13.1 Classes

Utilities for dealing with Python classes.

disseminate.utils.classes.**all_attributes_values**(*cls*, *attribute*)

Retrieve a list of all values for a given class's attribute for the given class and all its parent classes.

Parameters

cls [Type] The class object to inspect for parent classes.

attribute [str] The attribute whose value should be retrieved from each class.

Returns

parent_attributes [list] The list of all parent classe attributes.

Examples

```
>>> class A(object):
...     value = 'a'
>>> class B(A):
...     value = 'b'
...     value2 = (1, 2)
>>> class C(B):
...     value = 'c'
...     value2 = (2, 3)
>>> sorted(all_attributes_values(C, 'value'))
['a', 'b', 'c']
>>> sorted(all_attributes_values(C, 'value2'))
[1, 2, 3]
```

disseminate.utils.classes.**all_dicts**(*cls*)

Produce a dict with entries from a given class's __dict__ and the __dict__ of all its parent classes.

Precedence is given to the subclasses over parent classes if the attributes were overwritten by subclasses.

Examples

```
>>> class A(object):
...     a1 = 1
...     a2 = 2
>>> class B(A):
...     a2 = 3
...     b1 = 4
>>> d = all_dicts(B)
>>> d['a1']
1
>>> d['a2']
```

(continues on next page)

(continued from previous page)

```
3
>>> d['b1']
4
```

disseminate.utils.classes.**all_parent_classes**(*cls*)

Retrieve a list of all of parent classes for a given class.

Parameters

cls [Type] The class object to inspect for parent classes.

Returns

parent_classes [list] The list of all parent classes (not including the object class).

Examples

```
>>> class A(object): pass
>>> class B(A): pass
>>> class C(B): pass
>>> all_parent_classes(C)
[<class 'disseminate.utils.classes.B'>, <class 'disseminate.utils.classes.A'>]
```

disseminate.utils.classes.**all_subclasses**(*cls*)

Retrieve all subclasses, sub-subclasses and so on for a class

Parameters

cls [Type] The class object to inspect for subclasses.

Returns

subclasses [list] The list of all subclasses.

Examples

```
>>> class A(object): pass
>>> class B(A): pass
>>> class C(B): pass
>>> all_subclasses(A)
[<class 'disseminate.utils.classes.B'>, <class 'disseminate.utils.classes.C'>]
```

disseminate.utils.classes.**i_chain**

alias of `itertools.chain`

class disseminate.utils.classes.**weakattr**

Bases: `object`

A descriptor to store a weakref to an object attribute

13.13.2 Dict

Utility functions for dictionaries.

disseminate.utils.dict.**find_entry**(*dicts*, **keys*, *suffix=None*, *sep='_'*)

Search dictionaries for entries with string keys constructed from ordred combinations of the specified keys.

Dictionaries are searched in order of preference. String keys are constructed with parts constructed by joining the parts with the sep string.

Parameters

dicts [Union[dict, Tuple[dict], List[dict]]] One or more dictionaries to find entries in.

keys [str] One or more items to construct keys.

suffix [Optional[str]] If specified, try appending the given suffix to generate the combined key before trying the combined key itself.

sep [Optional[str]] The separator for joining key parts into a key.

Returns

value The value from the dict for the best match.

Raises

KeyError Raised if a key was not found.

Examples

```
>>> d = {'starter_middle1_term': 1,
...      'starter_middle1': 2,
...      'starter_middle2_term': 3,
...      'starter_middle2': 4,
...      'starter_term': 5,
...      'starter': 6}
>>> find_entry(d, 'starter', 'middle1', suffix='term')
1
>>> find_entry(d, 'starter', 'middle1')
2
>>> find_entry(d, 'starter', 'middle2', suffix='term')
3
>>> find_entry(d, 'starter', 'middle2' )
4
>>> find_entry(d, 'starter', 'term')
5
>>> find_entry(d, 'starter')
6
```

13.13.3 File

Utilities for manipulating files and paths

disseminate.utils.file.**link_or_copy**(*src*, *dst*)
Create a hard link, if possible, between a src filepath to a dst filepath, or copy if a link is not possible.

disseminate.utils.file.**parents**(*path*)
Generate a list of parent paths for the given path.

Parameters

path [str] A filepath or directory.

Returns

parents [List[str]] The parents of the path

Examples

```
>>> parents('/home/jlorieau/Documents/test.txt')
['/home/jlorieau/Documents', '/home/jlorieau', '/home']
>>> parents('tex/chapter1/chapter1.tex')
['tex/chapter1', 'tex']
```

13.13.4 List

Utilities for lists.

disseminate.utils.list.**chunks**(*lst*, *N*)
Return a generator in chunks of N.

Parameters

lst [Union[List, Tuple]] The iterable to break into chunks.

N [int] The size of chunks.

Returns

Generator A generator with sublists of items from 'l' broken into chunks of 'N'.

disseminate.utils.list.**dupes**(*lst*, *key=None*)
Find duplicates in a list using, optionally, a key mapping function.

Parameters

lst [list] The list to check for duplicates.

key [Optional[Callable]] A mapping function that takes a list item and returns an alternate value to compare for duplicates.

Examples

```
>>> dupes([1, 2, 2, 3, 3, 3, 4, 5, 6, 6, 7])
[2, 2, 3, 3, 3, 6, 6]
>>> dupes([(1, 'a'), (2, 'a'), (3, 'b'), (4, 'c')], key=lambda i:i[1])
[(1, 'a'), (2, 'a')]
>>> dupes([(1, 'a'), (2, 'a'), (3, 'b'), (4, 'c')], key=lambda i:i[0])
[]
```

class disseminate.utils.list.**filterfalse**

Bases: object

filterfalse(function or None, sequence) -> filterfalse object

Return those items of sequence for which function(item) is false. If function is None, return the items that are false.

disseminate.utils.list.**flatten**(*lst*)

Flatten a list of lists.

Examples

```
>>> list(flatten([1, 2, 3]))
[1, 2, 3]
>>> list(flatten([[1, 2, 3], [4, 5, 6]]))
[1, 2, 3, 4, 5, 6]
>>> list(flatten([[1,2], [3, [4, 5], 6]]))
[1, 2, 3, 4, 5, 6]
```

disseminate.utils.list.**md5hash**(*lst*)

Generates a hash of a list or tuple.

Examples

```
>>> md5hash(('a', 'b', 1))
'68b6a776378decbb4a79cda89087c4ce'
>>> md5hash(('a', 'b', '1'))
'68b6a776378decbb4a79cda89087c4ce'
>>> md5hash(['a', [1, 'b'], 'c'])
'1c81219649292a2ef9240fc997353078'
>>> md5hash(['a', ['1', 'b'], 'c'])
'1c81219649292a2ef9240fc997353078'
```

disseminate.utils.list.**transpose**(*lst*)

Transpose a list of lists

Examples

```
>>> transpose([[1, 2, 3], [4, 5, 6]])
[[1, 4], [2, 5], [3, 6]]
>>> transpose([[1, 2, 3], [4, 5]])  # Discard mismatched list entries
[[1, 4], [2, 5]]
>>> transpose([[1, 2], [4, 5, 6]])  # Discard mismatched list entries
[[1, 4], [2, 5]]
```

disseminate.utils.list.**uniq**(*lst*, *key=None*)

Find the unique items in the given list while preserving the order of the list and optionally using a key mapping function.

Parameters

lst [list] The list to check for duplicates.

key [Optional[Callable]] A mapping function that takes a list item and returns an alternate value to compare for unique entries.

Examples

```
>>> uniq([1, 2, 2, 3, 3, 3, 4, 5, 6, 6, 7])
[1, 2, 3, 4, 5, 6, 7]
>>> uniq([(1, 'a'), (2, 'a'), (3, 'b'), (4, 'c')], key=lambda i:i[1])
[(1, 'a'), (3, 'b'), (4, 'c')]
>>> uniq([(1, 'a'), (2, 'a'), (3, 'b'), (4, 'c')], key=lambda i:i[0])
[(1, 'a'), (2, 'a'), (3, 'b'), (4, 'c')]
```

disseminate.utils.list.**unique_everseen**(*iterable*, *key=None*)

List unique elements, preserving order. Remember all elements ever seen.

disseminate.utils.list.**unwrap**(*lst*)

Unwrap lists with only a single item.

Parameters

lst [list] The list to unwrap

Returns

unwrapped_list [list] The unwrapped list

Examples

```
>>> unwrap([1, 2, 3])
[1, 2, 3]
>>> unwrap([1])
1
>>> unwrap([[1]])
1
>>> unwrap([[[3]]])
3
```

13.13.5 String

String manipulation operations.

disseminate.utils.string.**convert_camelcase**(*string*, *sep='_'*)

Convert a camel-case string to a string to a lower-case string with a seperator character.

Parameters

string [str] The camel-case string to convert

sep [Optional[str]] The separator character.

Returns

processed_string [str] The converted string

Examples

```
>>> convert_camelcase('ProcessContext')
'process_context'
>>> convert_camelcase('processcontext')
'processcontext'
>>> convert_camelcase('ProcessContext', '.')
'process.context'
```

disseminate.utils.string.**find_basestring**(*strings*)

Evaluate the common base string amongst a list of strings.

Parameters

strings [List[str]] A list of strings

Returns

base_string [str] The common base string.

Examples

```
>>> find_basestring(['tester', 'test', 'testest'])
'test'
>>> find_basestring(['tester', 'test', 'testest', 'smile'])
''
```

disseminate.utils.string.**group_strings**(*lst*)

Group adjacent strings in a list and remove empty strings.

Parameters

lst [Union[list, str or *Tag*]] An AST comprising a strings, tags or lists of both.

Returns

processed_list The list with the strings grouped and cleaned up

Examples

```
>>> group_strings(lst=['a', 'b', '', 3, '', 4, 5, 'f'])
['ab', 3, 4, 5, 'f']
>>> group_strings(lst=['a', 'b', 'c', 3, 'd', 'e', 4, 5, 'f'])
['abc', 3, 'de', 4, 5, 'f']
```

class disseminate.utils.string.**groupby**(*iterable*, *key=None*) → make an iterator that returns consecutive
Bases: object

keys and groups from the iterable. If the key function is not specified or is None, the element itself is used for grouping.

disseminate.utils.string.**hashtxt**(*text*, *truncate=10*)
Creates a hash from the given text.

disseminate.utils.string.**nicejoin**(**items*, *sep=', '*, *term=' and '*)
Join a sequence of strings with a separator and an alternative terminal separator.

Parameters

***items** [Tuple[str]] A sequence of strings to join.

sep [Optional[str]] The separator to use for all items, other than the last item.

term [Optional[str]] The last separator to use in joining items.

Returns

joined_string [str] The joined string.

Examples

```
>>> nicejoin('Bill', 'Ted', 'Frances')
'Bill, Ted and Frances'
>>> nicejoin('Bill', 'Ted', 'Frances', term=' or ')
'Bill, Ted or Frances'
>>> nicejoin('Bill', 'Ted', 'Frances')
'Bill, Ted and Frances'
>>> nicejoin('Bill', 'Ted')
'Bill and Ted'
>>> nicejoin('Bill')
'Bill'
```

disseminate.utils.string.**replace_macros**(*s*, **dicts*)
Replace the macros and return a processed string.

Macros are simple string replacements from context entries whose keys start with the tag_prefix character (e.g. '@test'). The process_context_asts will ignore macro context entries that have keys which start with this prefix. This is to preserve macros as strings.

Parameters

s [str] The input string to replace macros within.

***dicts** [Tuple[dict]] One or more dicts containing variables defined for a specific document. Values will be replaced with the first dict found with that value.

Returns

processed_string [str] A string with the macros replaced.

Raises

MacroNotFound Raises a MacroNotFound exception if a macro was included, but it could not be found.

disseminate.utils.string.**slugify**(*text*, *entities=True*, *decimal=True*, *hexadecimal=True*, *max_length=0*, *word_boundary=False*, *separator='-'*, *save_order=False*, *stopwords=()*, *regex_pattern=None*, *lowercase=True*, *replacements=()*)

Make a slug from the given text. :param text (str): initial text :param entities (bool): converts html entities to unicode :param decimal (bool): converts html decimal to unicode :param hexadecimal (bool): converts html hexadecimal to unicode :param max_length (int): output string length :param word_boundary (bool): truncates to complete word even if length ends up shorter than max_length :param save_order (bool): if parameter is True and max_length > 0 return whole words in the initial order :param separator (str): separator between words :param stopwords (iterable): words to discount :param regex_pattern (str): regex pattern for allowed characters :param lowercase (bool): activate case sensitivity by setting it to False :param replacements (iterable): list of replacement rules e.g. [[‘|’, ‘or’], [‘%’, ‘percent’]] :return (str):

disseminate.utils.string.**space_indent**(*s*, *number=4*)

Indent a text block by the specified number of spaces.

Parameters

s [str] The string the indent

number [Optional[int]] The number of spaces to indent by.

Returns

indented_string [str] The indented string.

Examples

```
>>> space_indent('my test')
'    my test'
>>> t=space_indent("my block\n of text\nwith indents.")
>>> print(t)
    my block
     of text
    with indents.
```

disseminate.utils.string.**str_to_dict**(*string*, *strip_quotes=True*)

Parse a string into a dict.

Parameters

string [str] The string with dict-like entries separated by colons.

strip_quotes [Optional[bool]] If True, matched quotes (‘”) are stripped at the ends of value strings.

Returns

parsed_dict [dict] The parsed dict with keys and values as strings.

Examples

```
>>> string = '''
... first: one
... second:
...   multiline
...   entry
... third: 3
... '''
>>> d = str_to_dict(string)
>>> d == {'first': 'one', 'second': '  multiline\n  entry', 'third': '3'}
True
```

disseminate.utils.string.**str_to_list**(*string*)

Parse a string into a list.

Parameters

string [str] The string with list entries separated by semicolons, newlines or commas.

Returns

parsed_list [list] The parsed list with the string split into string pieces.

Examples

```
>>> str_to_list('one, two, three')
['one', 'two', 'three']
>>> str_to_list('one, 1; two, 2; three, 3')
['one, 1', 'two, 2', 'three, 3']
>>> str_to_list('''
... one, 1
... two, 2
... three, 3''')
['one, 1', 'two, 2', 'three, 3']
>>> str_to_list("friends")
['friends']
```

disseminate.utils.string.**strip_end_quotes**(*s*)

Strip matched quotes from the ends a string.

Parameters

s [str] String to strip matched quotes.

Returns

stripped_string [str] The same string with matched quotes striped.

Examples

```
>>> strip_end_quotes('This is my "test" string.')
'This is my "test" string.'
>>> strip_end_quotes('"This is my test string"')
'This is my test string'
>>> strip_end_quotes('"This is my test string "')
'This is my test string '
```

disseminate.utils.string.**strip_multi_newlines**(*s*)

Strip multiple consecutive newlines in a string.

Parameters

s [str] String to strip multiple consecutive newlines from.

Returns

stripped_string [str] The same string with multiple consecutive newlines replaced to single newlines.

Examples

```
>>> strip_multi_newlines("This is my\nfirst string.")
'This is my\nfirst string.'
>>> strip_multi_newlines("This is my\n\nsecond string.")
'This is my\nsecond string.'
```

disseminate.utils.string.**stub**(*s*)

Takes a string, like a docstring, and returns only the first line.

Parameters

s [str] The string to generate a stub for.

Returns

stub [str] The string with its first line stripped.

Examples

```
>>> stub(stub.__doc__)
'Takes a string, like a docstring, and returns only the first line.'
```

disseminate.utils.string.**titlelize**(*string*, *truncate=True*, *capitalize=False*)

Given a string, generate a condensed title.

```
>>> titlelize("My example caption. It has 2 sentences")
'My example caption'
>>> titlelize("My example caption. It has 2 sentences", capitalize=True)
'My Example Caption'
```

13.13.6 Tests

Utilities for testing.

disseminate.utils.tests.**strip_leading_space**(*string*, *no_spaces=4*)
Strip leading spaces at the start of newlines in a string.

13.13.7 Types

Generic types.

class disseminate.utils.types.**FloatPositionalValue**
Bases: *disseminate.utils.types.PositionalValue*

A placeholder for a positional *float* value in a dict.

class disseminate.utils.types.**IntPositionalValue**
Bases: *disseminate.utils.types.PositionalValue*

A placeholder for a positional *integer* value in a dict.

class disseminate.utils.types.**PositionalValue**
Bases: object

A placeholder for a positional value in a dict.

class disseminate.utils.types.**StringPositionalValue**
Bases: *disseminate.utils.types.PositionalValue*

A placeholder for a positional *string* value in a dict.

disseminate.utils.types.**ispositional**(*p*, *positionalvalue_type=<class 'disseminate.utils.types.PositionalValue'>*)
True, if the given parameter is a PositionalValue class or subclass.

Parameters

p [object] The parameter to test.

positionalvalue_type [*PositionalValue*] The PositionalValue class to test whether parameter is a subclass of this type.

Returns

bool True if parameter 'p' is the class or a subclass of PositonalValue (positionalvalue_type).

Examples

```
>>> ispositional('test')
False
>>> ispositional(IntPositionalValue)
True
>>> ispositional(PositionalValue)
True
```

disseminate.utils.types.**positionalvalue_type**(*p*)
Find the PostionalValue type for the given parameter.

Parameters

p [object] The parameter for which the PositionalValue class or subclass will be evaluated and returned.

Returns

positionalvalue_type [*PositionalValue*] The PositionalValue class or subclass that matches the given parameter.

Examples

```
>>> positionalvalue_type(3)
<class 'disseminate.utils.types.IntPositionalValue'>
>>> positionalvalue_type('test')
<class 'disseminate.utils.types.StringPositionalValue'>
>>> positionalvalue_type('23')
<class 'disseminate.utils.types.IntPositionalValue'>
>>> positionalvalue_type('3.23')
<class 'disseminate.utils.types.FloatPositionalValue'>
>>> positionalvalue_type('src/media/image.png')
<class 'disseminate.utils.types.StringPositionalValue'>
```

CHAPTER

FOURTEEN

RELEASE NOTES

14.1 v2.3

1. *Webserver*. Implementation of the Tornado library for the internal preview webserver.

14.2 v2.2

1. *CLI*. Implementation of the cli *init* subcommand.

14.3 v2.1

1. *Epub*. Implementation of epub version 3 as document target format.
2. *Documentation*. Documentation updates for API changes, added an Overview section with Installation, Quickstart and Features sections.

14.4 v2.0

1. *Builders*. Implementation of thread locking and ThreadPoolExecutor for builder threads.
2. *Builders*. Implementation of process timeouts.
3. *Templates*. CSS updates to `books/tufte`, `books/novel`.

14.5 v1.0

1. *Builder*. Implementation of the builder framework for multithreaded and multiprocess compilation of documents and document dependencies.
2. *Dependency Manager*. Elimination of the dependency manager, which is now replaced by builders.

14.6 v0.21

14.6.1 Tags

1. *Problem boxes*. Added a feature box for problems.

14.7 v0.20

14.7.1 Tags

1. *Tables*. Tags to render tables.
2. *Data*. Tags to incorporate data into tables or plots.
3. *Captions*. Captions now render in html using the `caption` or `figcaption` tags.
4. *Figures*. Figures now render in html using the `figure` tag.
5. *Equations*. Refactor image, equation and paragraph tags to automatically identify equations in a block paragraph, without specifying the equation environment (ex: `@eq[env=align*]{...}`)

14.8 v0.19

14.8.1 Tags

1. *Feature Boxes*. A box with a feature for a text, like an example box (`@examplebox`) or a general featurebox (`@featurebox`)
2. *Headings*. The behavior of heading tags to not list the contents of the tag by default. Instead, a label tag is created, and the content of the tag is included if a `@label.title` macro was included.
3. *Equations*

   3.1. *tex*: Block equations (equations in their own paragraph) are written without newlines before/after the environment in latex.

14.8.2 Templates

1. *books/tufte*. Remove indentation before captions.

14.8.3 Documentation

1. *Feature Boxes*. Add the `@featurebox` and `@examplebox` tags to the language documentation.

14.8.4 Bug Fixes

1. Fixed the renaming of filenames with periods in converters. Created a new path utility, `rename`, that more intelligently renames files.

14.9 v0.18

14.9.1 Document

1. Add local path (parent dir) to the search paths for each document
2. Files are copied both from the local path of a document or subdocument or the project root path.

14.9.2 Converters

1. *Imagemagick*. Implemented a `.tif`/`.tiff` converter to `.png`

14.9.3 Bug Fixes

1. Wrap filenames and paths in curly braces for LaTeX `\includegraphics` command. This is needed to deal with filenames that include special characters, like periods.

14.10 v0.17

14.10.1 Tags

1. Implementation of the `@list` and `@outline` tags.
2. Implementation of `@prev` and `@next` tags with signals
3. Implementation of a general `html_list` function for `@list` and `@toc` tags

14.10.2 Document

1. Implementation of signals.

14.11 v0.16

14.11.1 Backend

1. *Webserver*. Implement sanic for preview server.
2. *Heritable tags*. Allow tags to be copied and inherited between documents. This enables `@toc` tags and navigation tags to be shared between a document and its subdocuments.
3. *Navigation tags*. Add navigations tags `@prev` and `@next` to add html links for the previous or next page. These are available as the `prev` and `next` entries in the context.

14.12 v0.15

14.12.1 Interfaces

1. *Click*. Implemented Click for the CLI
2. *Flask*. Removed the server interface from the project

14.13 v0.14

14.13.1 Interfaces

1. *Flask*. Implement Flask web server for viewing the document list and rendered products.

14.13.2 Backend

1. *BaseContext*. Refactor to copy values from parent_context, rather than use a ChainMap implementation. This significantly speeds up look ups and loading of documents.
2. *BaseContext*. Create shallow copies of mutables from the parent_context so that these aren't modified directly. This is only done for mutables that have `copy` methods so that some mutables, like the `LabelManager` and `DependencyManager`, can be preserved by all contexts and documents in a project
3. *ProcessContextHeader*. Refactor to use the new implementation of the BaseContext. The `ProcessContextHeader` now does the work of loading templates and loading additional context headers.
4. *Renderers*. Simplify the API.

14.14 v0.13 beta

14.14.1 Backend

1. *Documentation*. Update documentation to use sphinx.
2. *Context*. Rewrote context to work like a ChainMapping with inherited entries from a parent_context.
3. *Paths*. Allow relative links and urls.
4. *TemplateStrings*. Eliminated the TemplateString class with a replace_macro function.
5. *Equation Tags*. Implement a new pdf cropping converter to more cleanly crop equation images in targets like `.html`.
6. *Attributes*. Refactored tag attributes to use an ordered dict instead of tuples. The Attributes class now includes useful utility functions, like filter and exclude.
7. *Formats*. Refactor the formatting of targets for tags with a new formats sub-module. This module now checks for allowed tags in the settings. The formats submodule also isolates the dependency of external packages, like lxml, to one place instead of multiple places.
8. *Processors*. Created a ProcessorABC abstract base class as a chain of command class for objects like tags and context. Included a simple listing of processors in the CLI.

9. *Tags*. Eliminate the ast submodule and replaced with a TagProcessor.
10. *Document*. Moved context processors to the document submodule and refactor to use the ProcessorABC.
11. *Label Manager*. Refactored to simplify the assignment of labels, the resetting of label counters and to minimize the dependency of labels for tags. Also added a set of label processors based on the ProcessorABC.

CHAPTER

FIFTEEN

INDEX

PYTHON MODULE INDEX

d

INDEX

Symbols

A

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X