
Documentation Standard

Release 0.0

Wolfgang Scherer

Mar 19, 2023



The famous pipe. How people reproached me for it! And yet, could you stuff my pipe? No, it's just a representation, is it not? So if I had written on my picture 'This is a pipe', I'd have been lying!

—René Magritte[TORC]

No software survives contact with the user.

—not Helmuth von Moltke the Elder

Quickstart

Single shot standalone document:

```
sda chapter new 'Doc Title'          # => README-doc-title.txt

sda readme README-doc-title.txt      # => README-doc-title.html
sda readme -f p README-doc-title.txt # => README-doc-title.pdf
sda readme --format pdf README-doc-title.txt # => README-doc-title.pdf
```

Book w/chapter documents:

```
sda init "Project Title"            # => README.txt + doc/
sda chapter add "Title"             # => add chapter README-title.txt
sda automodule MODULE               # => module documentation from template

sda make html                       # => doc/_build/html/
sda make latexpdf                   # => doc/_build/latex/

sda view html                       # xdg-open 'doc/_build/html/index.html'
sda view pdf                        # xdg-open "$( echo doc/_build/latex/*.aux | sed 's,\.aux
↪$,\.pdf,;1q' )"

sda make clean                      # remove generated files => rm -rf doc/_build/
                                    # (necessary, when e.g., new chapters are added)
```

Update sphinx-doc framework for book documents (requires manual resolution!):

```
sda update                          # => book/README.txt + book/doc (re-created for update)
```

Find sphinx-doc directories:

```
sda locate --home
sda locate --sed SCRIPT

sda locate --grep [OPTIONS] [PATTERN]
```

Abstract

- Documentation is maintained in the source code and in ASCII files named `README.txt` or `README-topic.txt`, since binary documentation (Word, Visio) that requires a graphical user interface for viewing/editing is extremely counterproductive (See section 8, Relevance of Documentation).
- `reStructuredText` is used as markup language. It also happens to be the official markup language for the Python programming language.
- `Sphinx` extends the roles and directives of `reStructuredText`. It converts documents and Python doc strings to various formats such as HTML, PDF and EPUB. (See section 14, Sphinx Documentation Generator).
- UML diagrams are used to visualize program design and script algorithms. (See section 12, Unified Modeling Language).
- `PlantUML` is the program of choice for converting textual diagram descriptions into SVG, PNG or PDF images. (See local copy of `PlantUML Language Reference Guide`).
- **!todo!** Integrate `Lightbox2` to display full version of images.

```
| README-redistribution-of-info-items.txt | * /home/sw/project/  
↔administration | #mv# /home/da/project/documentation |
```

CONTENTS

List of Figures	iv
List of Tables	v
List of Code Blocks	vi
1 Introduction	1
2 Sphinx Doc	2
2.1 Documentation workflow	2
2.2 Initialize documentation	2
2.3 Documentation output formats	4
2.4 Update documentation template structure	4
2.5 Document administration	6
2.6 Structural specification	10
2.7 How to properly move chapter files and sections	16
2.8 ReST section overlines	17
3 sda chapter new is Faulty	18
3.1 From Gibberish to Brilliant Clarity	18
4 Figures	21
4.1 Figures with numbers	21
4.2 See also	25
4.3 Check README for figure requirements	25
4.4 Templating with automatic labels (NO!)	26
5 Citations	28
6 Glossary	29
6.1 As for the style of glossary entries	29
6.2 Multiple <i>glossary</i> directives	29
6.3 Combinations of glossary and abbreviation generation	30
7 UML annotations - line_diversion	36
7.1 Emacs support	36
7.2 Annotation tags and markers	36
7.3 Practical annotation	38
7.4 Activity Diagrams (extracted)	40
7.5 Class Diagram (extracted)	44
7.6 Command/Module Documentation	45
8 Relevance of Documentation	54
8.1 Introduction	54
8.2 Source and documentation management	55
8.3 Minimum Distance to Source Code	57

8.4	Inertia	57
8.5	Single Editor for Source and Documentation	59
8.6	Synchronization	59
8.7	Battling Human Inertia	62
9	Version Control System	65
9.1	Cherry-Picking	65
10	VCS - Mercurial	67
10.1	Repository Manipulation	67
11	VCS - Git	68
11.1	GUI	68
11.2	github fork	68
11.3	Quickstart	69
11.4	Tricks	70
11.5	Create branches	72
11.6	Patches	73
11.7	Github Forking	73
11.8	Resolving merge conflicts with git and kdiff3	74
12	Unified Modeling Language	75
12.1	UML Introduction	75
12.2	UML Diagrams	77
12.3	PlantUML	88
12.4	yUML	90
12.5	Other UML Tools	92
12.6	Summary	97
13	Tools	99
13.1	Document Generation Issues	99
13.2	Activity Diagrams for rst2md.sh	100
13.3	Activity Diagrams for sphinx-doc-locate.sh	101
13.4	Activity Diagrams for bin/inst.sh	102
14	Sphinx Documentation Generator	106
14.1	ReStructuredText Tips and Issues	106
14.2	Slides	108
14.3	Sphinx Themes	110
14.4	Graphviz Dot	110
14.5	Sphinx Mercurial	111
14.6	ReStructuredText and Sphinx bridge to Doxygen	112
15	Diagram generators	113
16	Emacs vs. Vi vs. Eclipse vs. anyIDE	114
16.1	How to install latest stable Emacs in Ubuntu	114
16.2	Point, mark, region, kill ring	116
16.3	Undo	118
16.4	Abbreviations	119
16.5	Dynamic Abbreviation Expansion	120
16.6	Key Sequences	121
16.7	Extensions	121
16.8	Tips and Tricks	121
16.9	Useful packages	121
16.10	Symbol tags	124
16.11	Directory/filename shortcuts	125
17	Document Snippets	127
17.1	Document Snippet Definition	127

17.2	Snippet Tag Substitutions	129
17.3	Document Snippet References	129
17.4	Replacement Facilities	132
17.5	sphinx_doc_snip.py	132
18	Snippets	144
18.1	Templating	144
19	High Contrast Colors	145
19.1	High Contrast Palette with 24 Colors	145
19.2	Thunderbird Tags	145
19.3	High Contrast Palette with alternative X11 colors	146
19.4	References	146
20	X11 Colors	147
20.1	X11 Colors - Web Colors	147
20.2	X11 Colors	149
20.3	X11 Colors - Gradients	154
20.4	X11 Colors - Grey Scale	163
21	Knowledge Organization	166
21.1	Trees	166
21.2	Generalization	171
21.3	Conclusion	176
22	Scratch	178
22.1	Style Guide	178
22.2	Emacs	178
22.3	Other Diagrams	179
22.4	Activity Diagrams	179
22.5	Emacs Buffers with Highlighting	180
22.6	Tastatur- und Spracheinstellung	181
22.7	Build statistics	181
23	Questions	183
24	rst-mode etags Support	187
24.1	Resources	187
24.2	etags Interface	187
24.3	diffmap	189
	Abbreviations	197
	Glossary	198
	Bibliography	199
	Python Module Index	200
	Index	201

LIST OF FIGURES

2.1	Documentation workflow	3
2.2	SphinxDoc - update template structure	5
2.3	SphinxDoc - compare updated files with original tree	5
4.1	Numeric references example	21
4.2	Figure example Magritte's Pipe	22
4.3	UML test	24
4.4	dot test	25
4.5	Fig Ctr Caption	25
6.1	Combinations of Glossary and Abbreviation Generation	31
6.2	Generated output files for glossary terms enabled	31
6.3	Generated output files for glossary terms disabled	32
6.4	Glossary class	33
6.5	Glossary parser activity diagram	33
6.6	Activity diagram for parsing a file	34
6.7	Glossary parser state diagram	35
8.1	I think they might be having sushi	55
8.2	Source and documentation management	56
8.3	Guaranteed and preferred editors	56
8.4	Sunny day documentation update	58
8.5	Rainy day documentation update	58
8.6	Documentation update decision	59
8.7	Documentation Update Decision With Single Editor	60
8.8	Editing with single editor	61
8.9	Geeks and repetitive tasks	63
8.10	Using Automation to assist	64
12.1	Big Brother is watching you	81
12.2	Caption with bold and <i>italic</i>	82
12.3	Activitiy of the day	85
24.1	Correlate Line Numbers from Diff Output	190
24.2	Correlate Line Numbers from Diff Output	193
24.3	Correlate Line Numbers from Diff Output	196

LIST OF TABLES

4.1	Emacs support functions for generating labels	21
4.2	single table cell (rectangle)	23
16.1	Mark line block between two symbol tags	125

LIST OF CODE BLOCKS

4.1	option <i>numfig</i> in <code>conf.py</code>	21
4.2	Figure example Magritte's Pipe	22
4.3	<code>table</code> directive with single table cell (rectangle)	23
4.4	Definition of a general code-block listing	23
4.5	Example with Elisp code	23
4.6	Some Emacs Lisp code	24
4.7	<code>uml</code> directive for <code>plantuml(1)</code> diagrams	24
4.8	<code>graphviz</code> directive for <code>dot(1)</code> graph diagrams	24
4.9	<code>figctr</code> directive	25
16.1	Basic symbol tag navigation	124
24.1	workspace register setup	187
24.2	check for diff edge case empty file	195
24.3	check for diff edge case empty file, diff output	195

INTRODUCTION

The documentation framework described in [section 2, Sphinx Doc](#), supports single file articles and books with chapters and fully configurable build environment. With a separate doc directory, a subset of chapters can be built as a book part.

The documentation is processed with the [Sphinx](#) document generator to produce various output formats (see [section 14, Sphinx Documentation Generator](#)).

The theory behind *Sphinx Doc* is discussed in [section 8, Relevance of Documentation](#).

The reasons for advanced concepts like document snippets, which expand the traditional knowledge collection beyond simple cross referencing (hyperlinks), are discussed in [section 21, Knowledge Organization](#).

SPHINX DOC

Sphinx Doc is a collection of scripts and templates to allow for simple setup, administration and output formatting of `reStructuredText` documents.

The focus is on providing a simple standalone README article and a book with chapters made from single README files.

It employs `snippets(1)` and the python package `PyJsMo`.

2.1 Documentation workflow

[figure 2.1](#) shows the definitive state diagram for the documentation workflow. Some state transitions are triggered by a list of mandatory shell commands. For alternatives, e.g.,

$$\left\{ \begin{array}{l} \text{touch file} \\ \text{sda automodule module} \\ \text{sda chapter add 'Chapter title'} \end{array} \right.$$

at least one command or an equivalent must be executed. **There is no exception to this requirement.**

2.2 Initialize documentation

2.2.1 New article

The command

```
sda chapter new 'Chapter title'
```

is roughly equivalent to

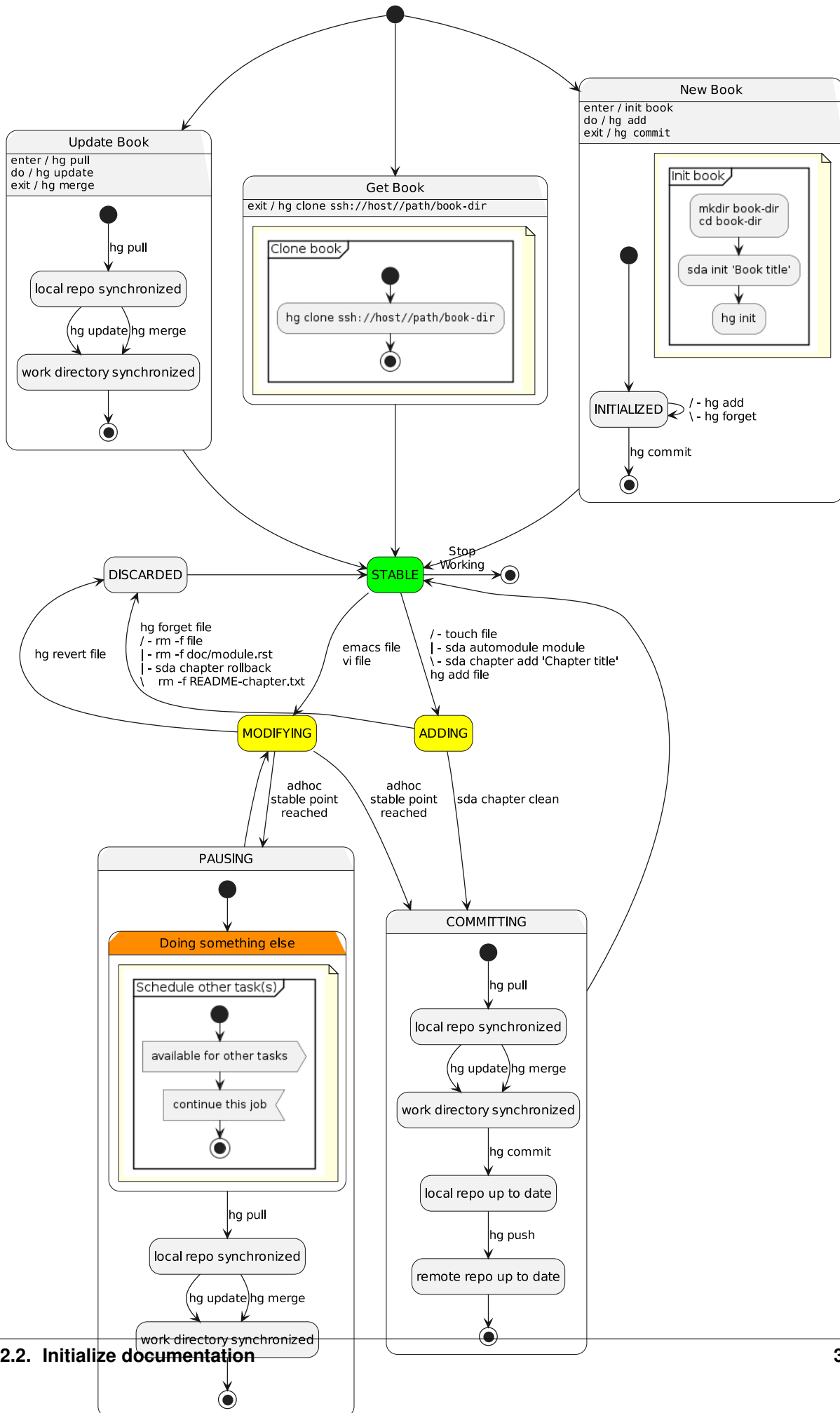
```
snn --mode rst --key title --value 'Chapter title' README-chapter-title.txt
```

2.2.2 New book

- Make book directory
- `cd` to book directory
- Execute

```
sda init "Book Title"
```

This creates an empty `README.txt` and a `doc` subdirectory for building the full version of a documentation book with chapters.



2.3 Documentation output formats

2.3.1 Article format

```
sda readme README.txt # => README.html
sda readme --format pdf README.txt # => README.pdf
```

2.3.2 Book format

```
sda make html # => doc/_build/html/
sda make latexpdf # => doc/_build/latex/

sda view html # xdg-open 'doc/_build/html/index.html'
sda view pdf # xdg-open "$( echo doc/_build/latex/*.aux | sed 's,\.aux
↪$,\.pdf,;lq' )" )"
```

2.4 Update documentation template structure

!todo: add to glossary

1. The **book directory** must not have any uncommitted changes.
2. The **book directory** must be synchronized with master repository.
3. The book at the **book directory** should build without warnings.

If there are warnings, record them to be able to distinguish between old warnings and new warnings.

```
sda make clean
sda make html
sda make pdf
```

4. Update template framework (snippets) at the **book directory**

```
sda update
```

This creates a new sub-directory `BOOK-DIR-update` with a new documentation project using the title from the main README (see [figure 2.2](#)).

Any generated files in `BOOK-DIR-update`, that are not present in `BOOK-DIR` updated, are moved to the corresponding location. Files that are identical (ignoring copyright lines) are deleted. (see [figure 2.3](#)).

5. Files in `BOOK-DIR-update` that differ from files in `BOOK-DIR` must be merged manually with `M-x ediff-directories` or `kdiff3(1)`.

```
(let* ((project (file-name-base (directory-file-name default-directory)))
      (update (concat default-directory project "-update")))
      (ediff-directories update default-directory ""))
```

6. Run a fresh build to detect any errors.
7. Commit changes using the phrase *template framework update* (don't forget adding new files).
8. Push commits to master repository.
9. Publish repository.
10. Remove update directory.



figure 2.2: SphinxDoc - update template structure

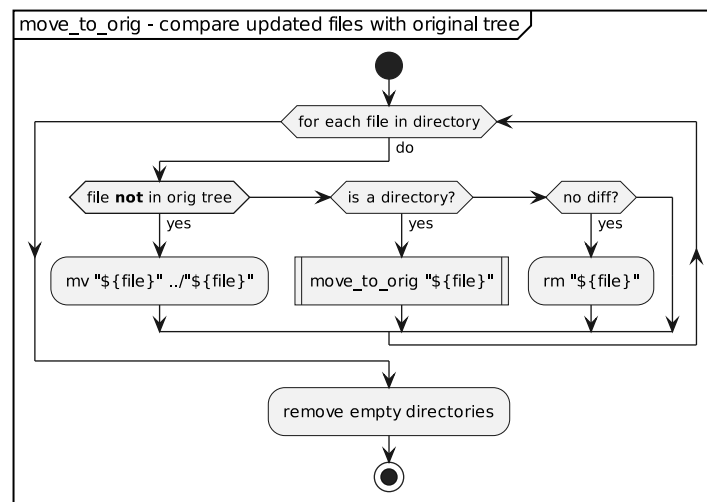
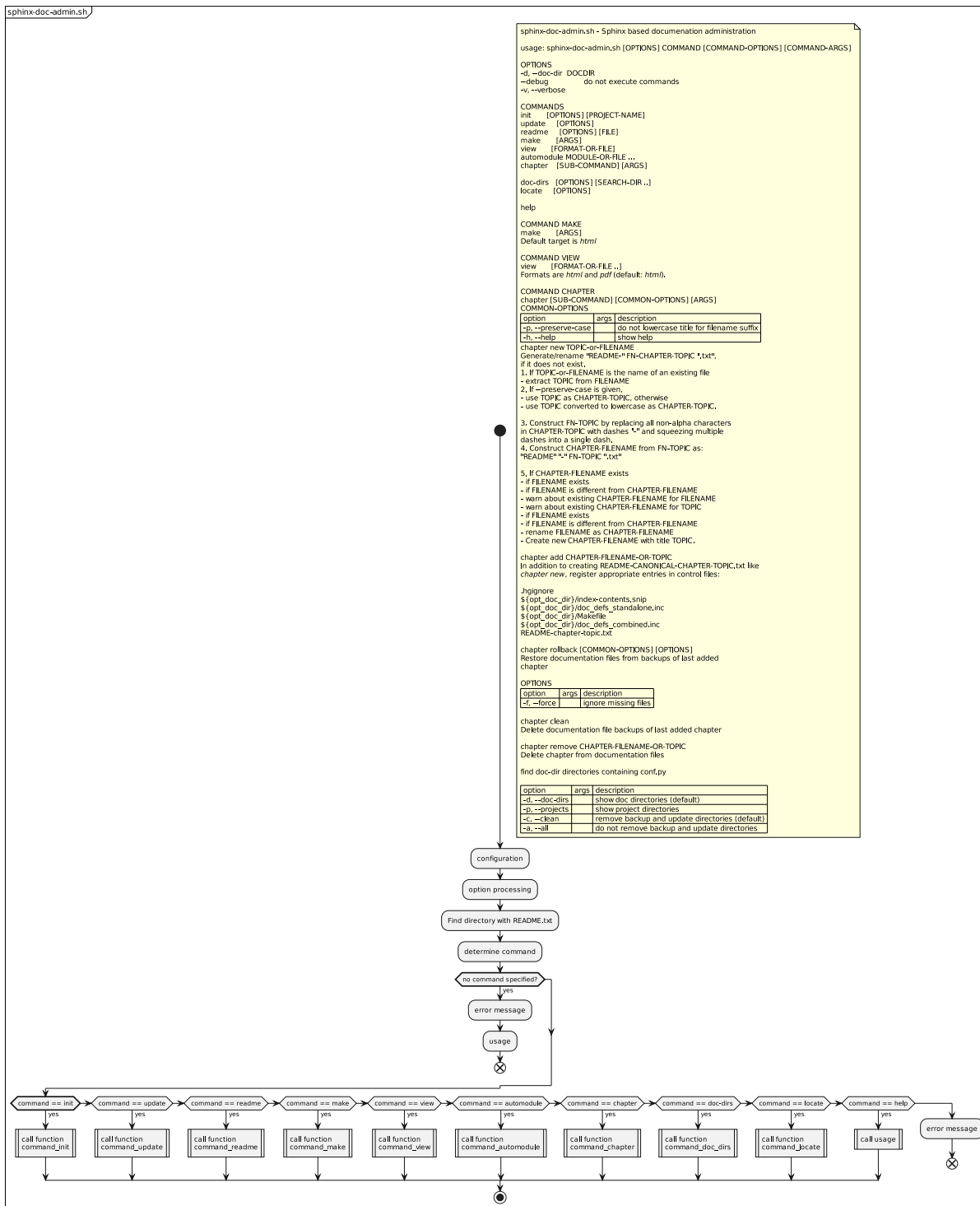


figure 2.3: SphinxDoc - compare updated files with original tree

2.5 Document administration

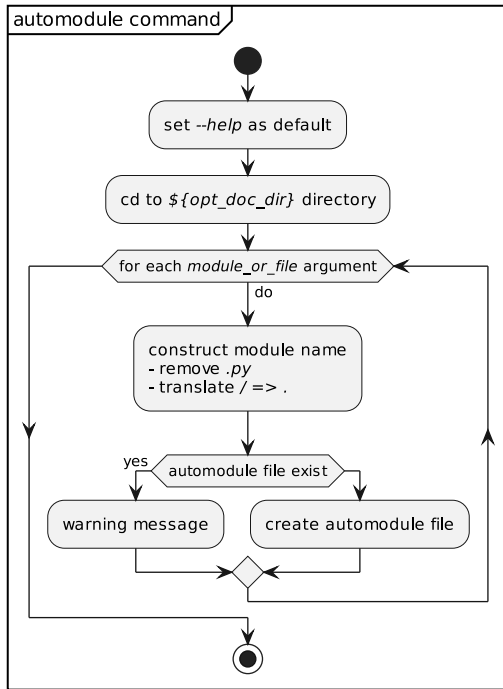
Document administration is performed with `sda`, a shortcut link to `sphinx-doc-admin.sh`.



2.5.1 Automodule files

Add automodule documentation files for python modules to the `doc` directory.

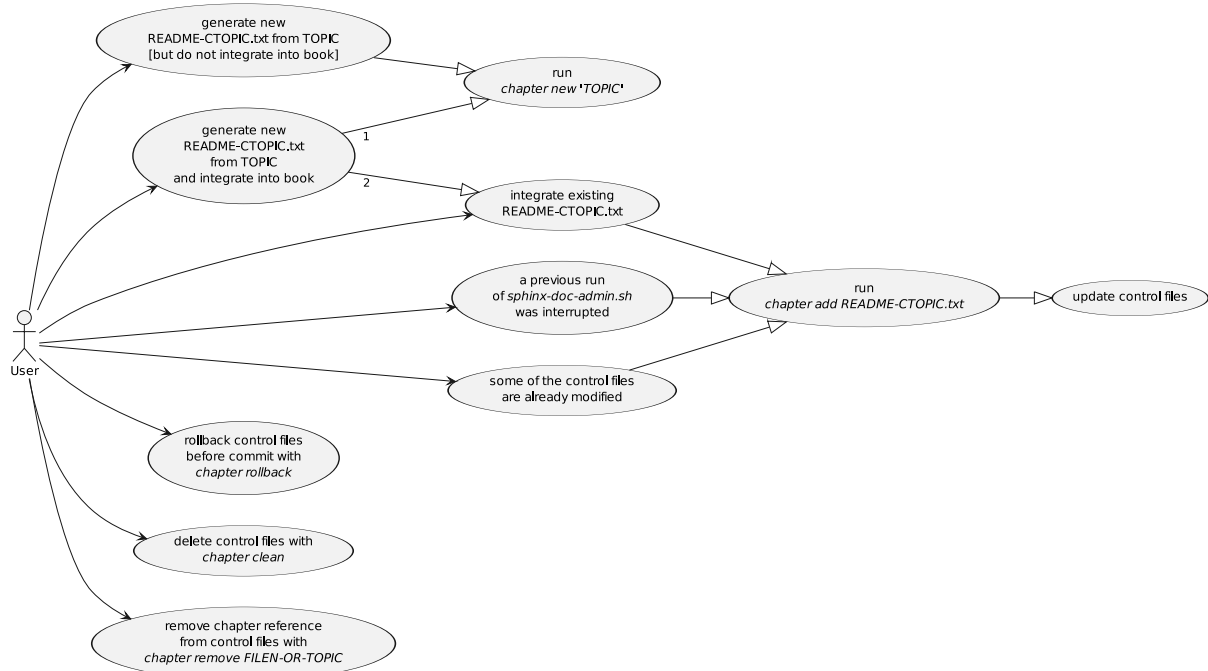
```
sda automodule MODULE-OR-FILE ...
```

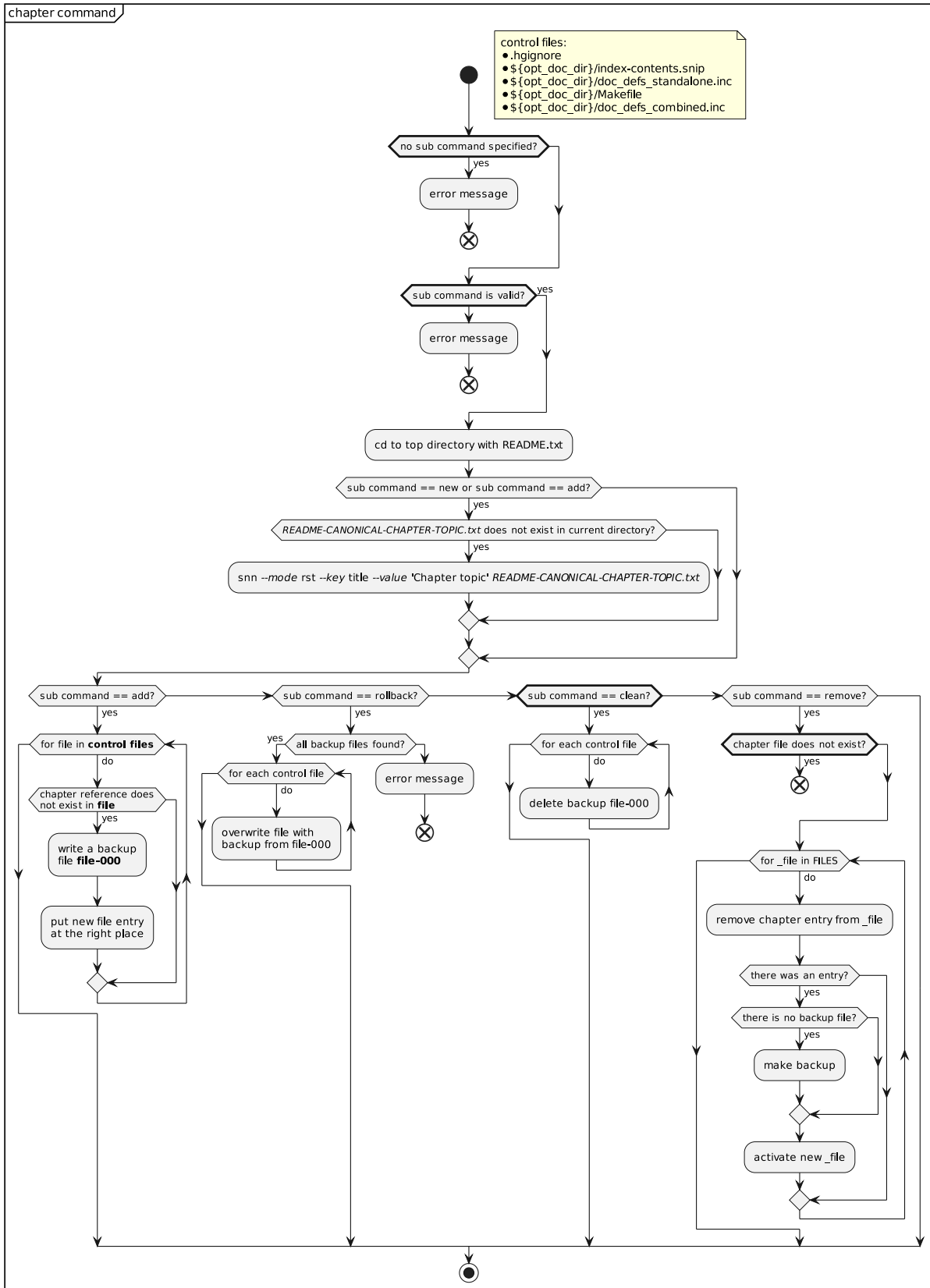
2.5.2 Chapters

Latest versions of the documentation have `!chapter:` symbol tags, which can be easily found with `M-x symbol-tag-grep-find` (usually defined as `M-g`).

Chapter administration entails a number of use cases:



```
sda chapter SUB-COMMAND
```



Adding a chapter

The following steps are automatically performed with:

```
sda chapter add topic
```

- Create a new README-topic.txt:

```
snn --mode rst README-topic.txt
```

- Add a new entry to the variable **CHAPTERS** in doc/Makefile:

```
CHAPTERS += ../README-topic.txt
```

- Add the chapter file base topic to the toctree directive in doc/index-contents.snip:

```
.. toctree::
   :maxdepth: 1

   topic
```

Add a rule to .hgignore:

```
^doc/topic\.rst\.auto$
```

Define a document reference in doc/doc_defs_standalone.inc:

```
.. |chapter-topic|          replace:: document :file:`README-topic`
```

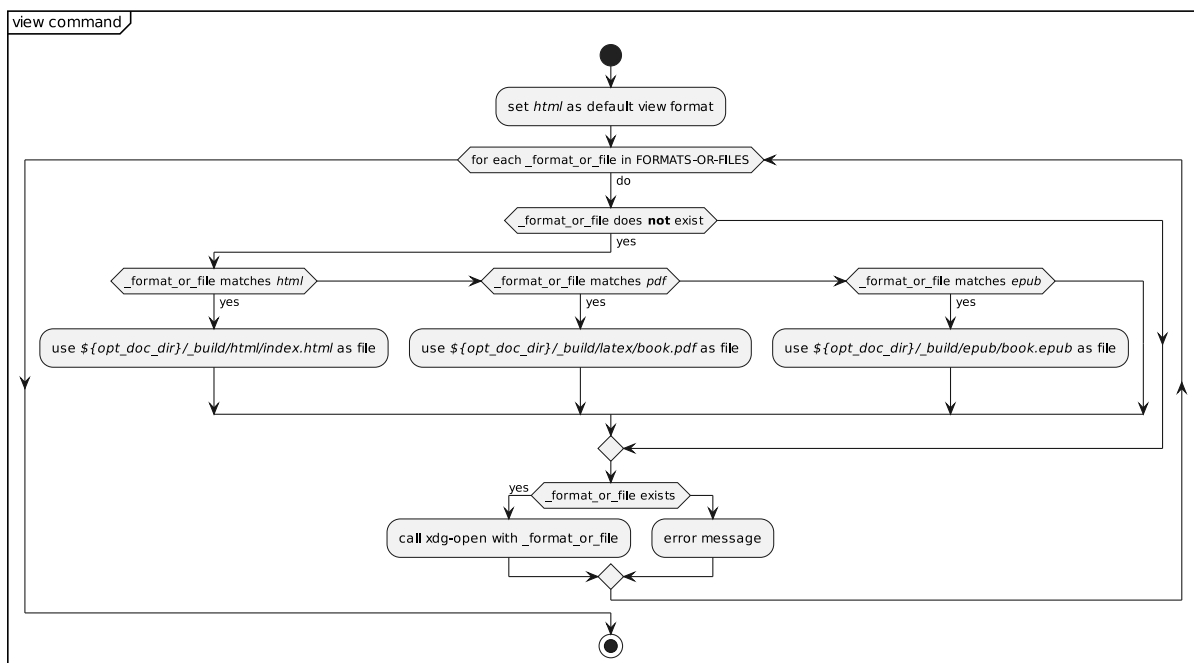
Define a chapter reference in doc/doc_defs_combined.inc:

```
.. |chapter-topic|          replace:: chapter :doc:`topic`
```

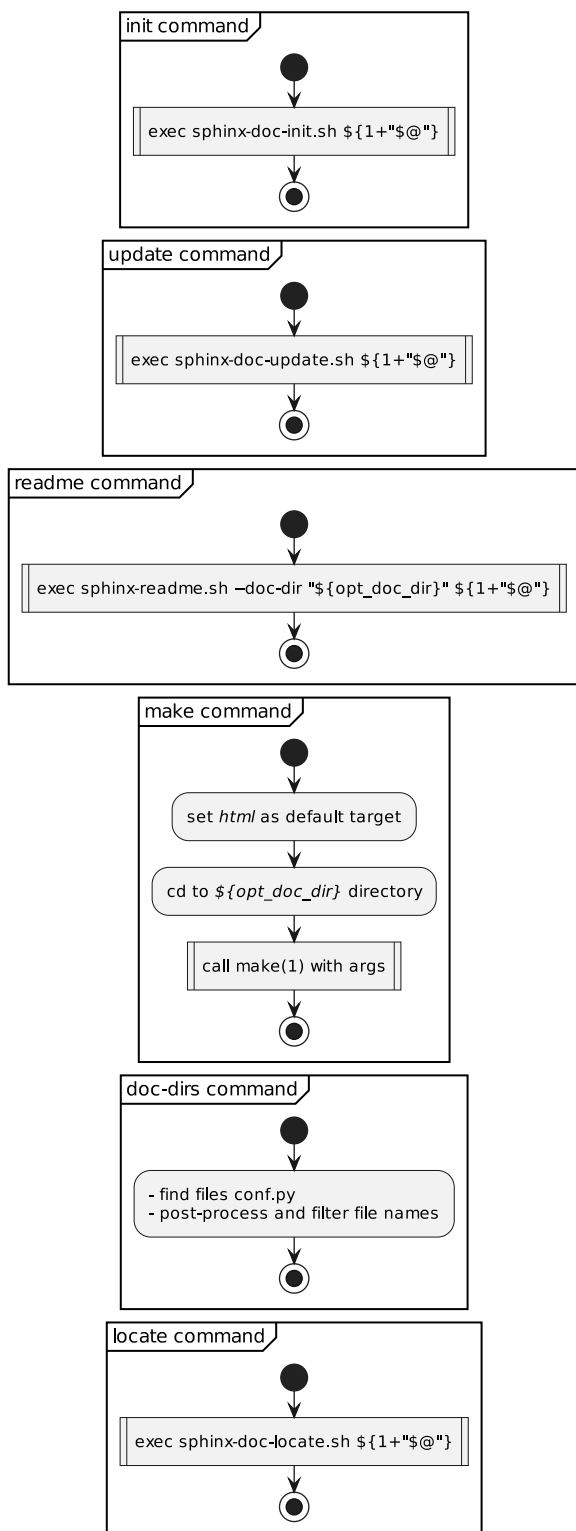
2.5.3 View processed documents

```
sda view FORMAT-OR-FILE
```

```
sda view html
sda view pdf
```



2.5.4 Command handlers for backends

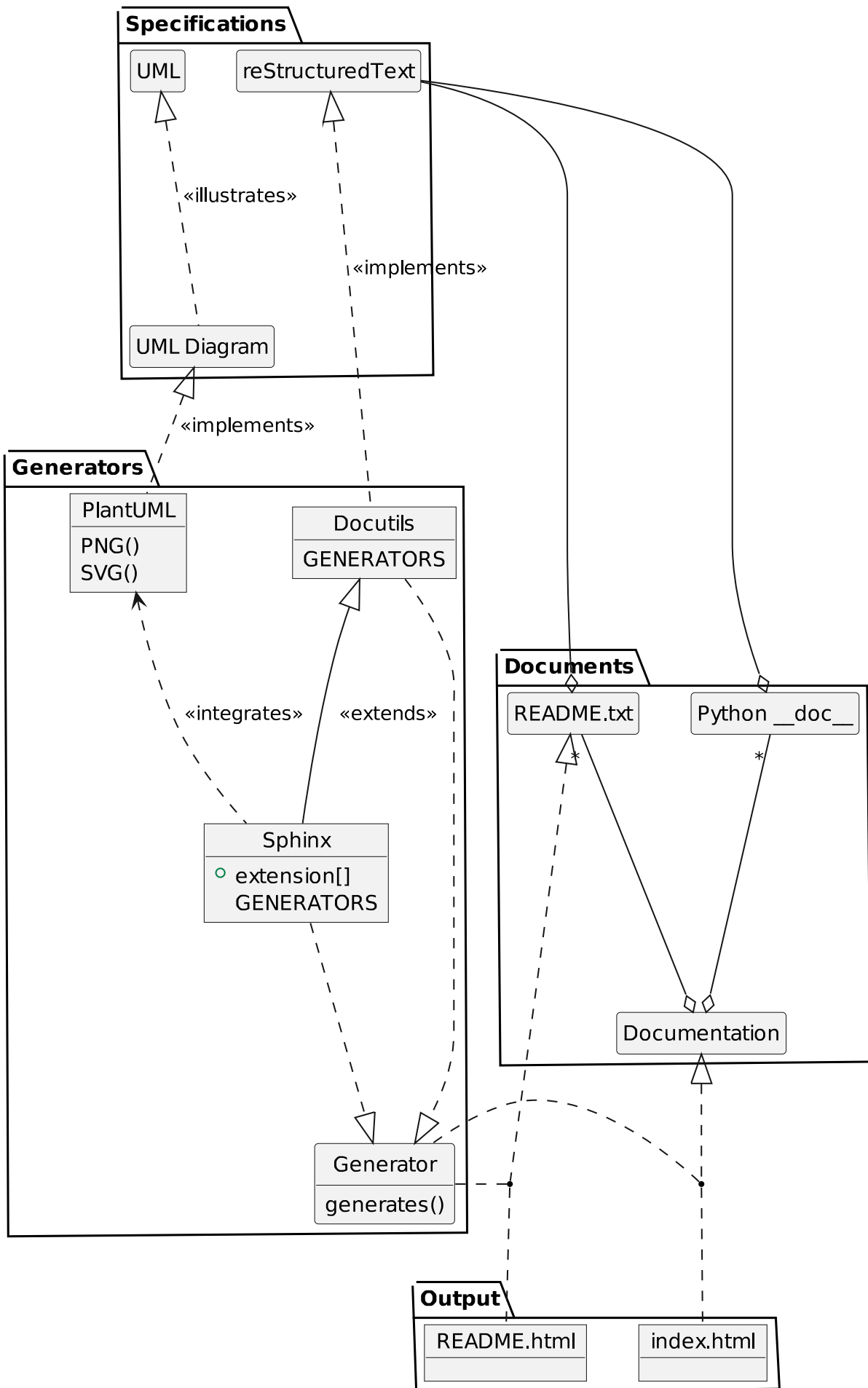


2.6 Structural specification

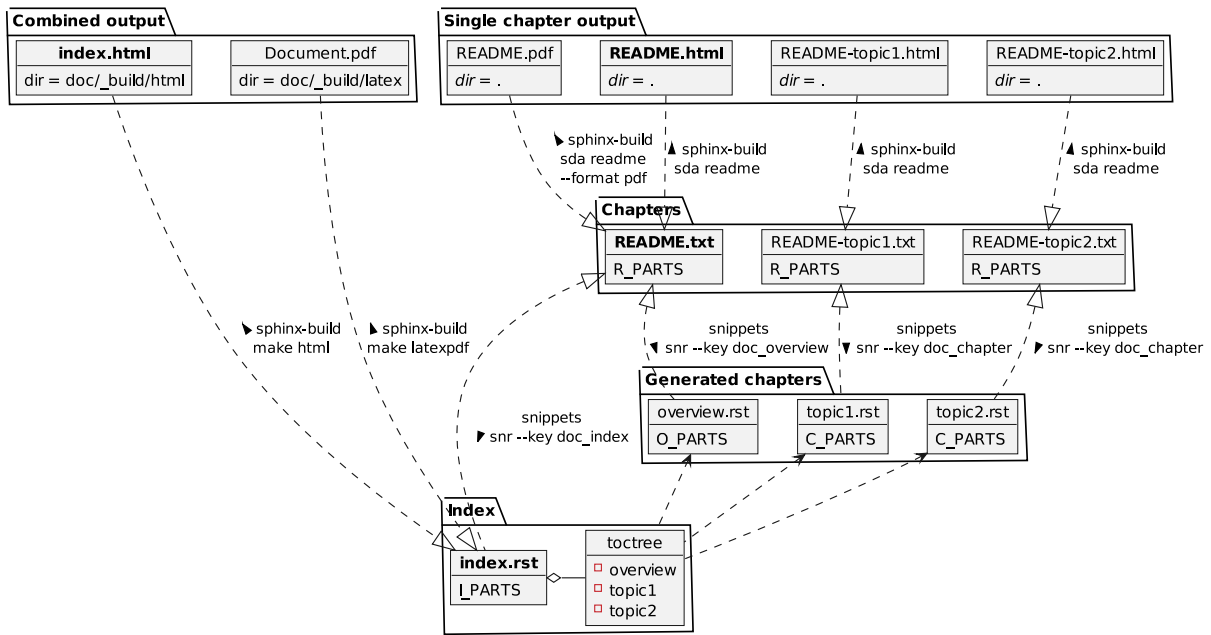
- `reStructuredText` is a markup language specification in the class of wiki markup languages. (Examples for other markup languages are HTML, TeX/LaTeX, NROFF). `reStructuredText` is the official Python documentation markup.

- [UML](#) is used for documenting programs. It is created from textual descriptions with [PlantUML](#). See [section 12, Unified Modeling Language](#) for details.
- [Docutils](#) is the reference library for translating [reStructuredText](#) markup into other formats like HTML/PDF/EPUB.
- The document generator [Sphinx](#) extends [Docutils](#) and the [reStructuredText](#) specification with roles and directives. There are also various extensions to integrate other markup specifications (e.g. [PlantUML](#), [Pygments](#) for highlighting). See [section 14, Sphinx Documentation Generator](#) for further details .

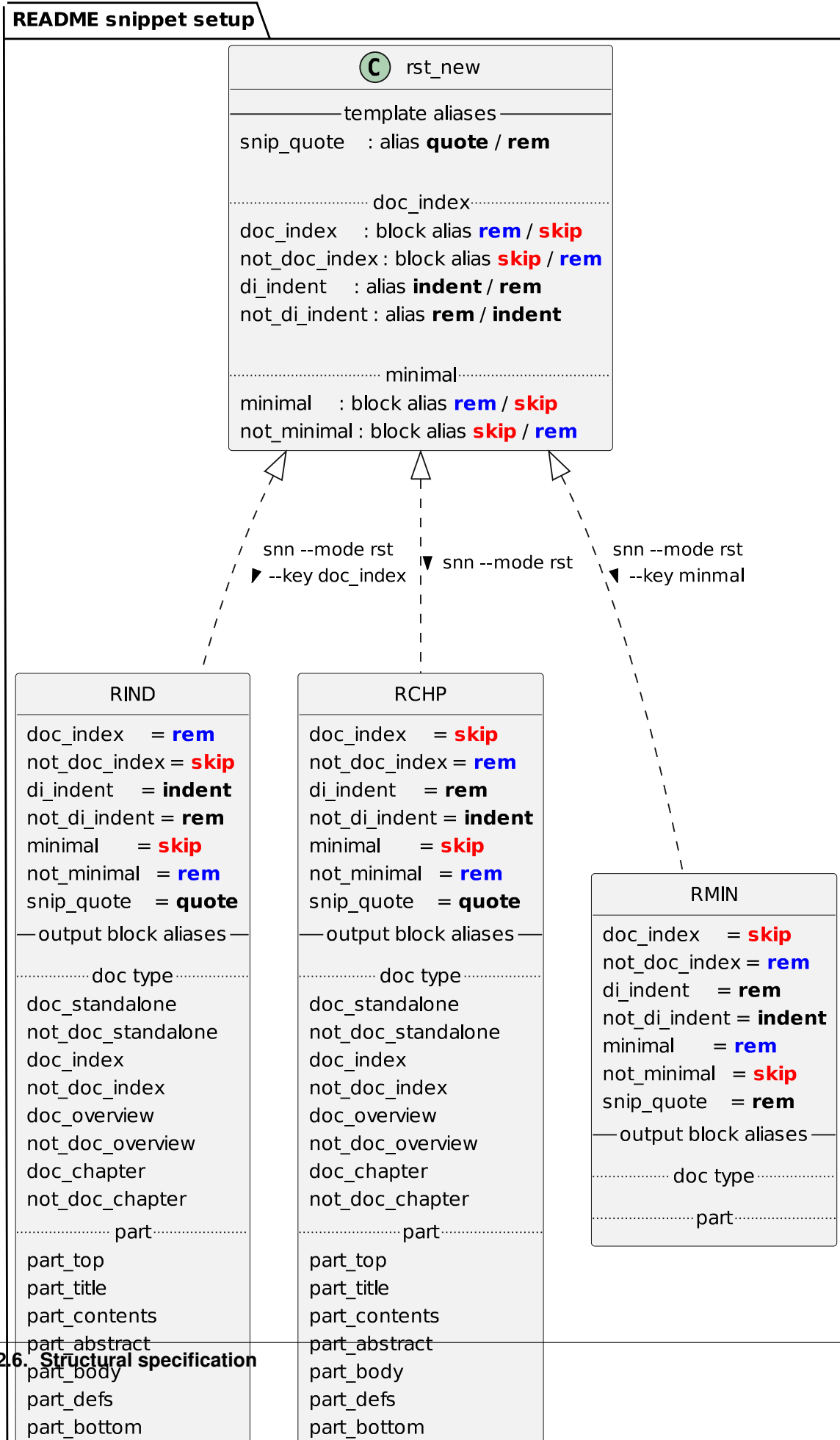
The program `sphinx-build(1)` generates HTML/PDF/EPUB documentation from [reStructuredText](#) documents and doc strings of `python(1)` modules. It is also used in the command `sda readme` (`sphinx-readme.sh`) to generate standalone HTML/PDF/EPUB from a single README file.



2.6.1 README chapters

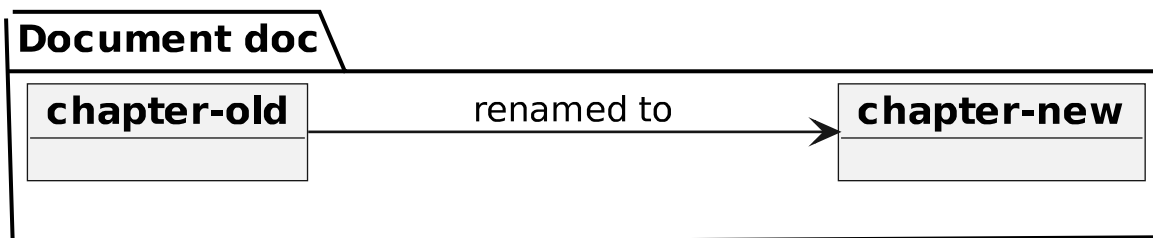


2.6.2 README snippets



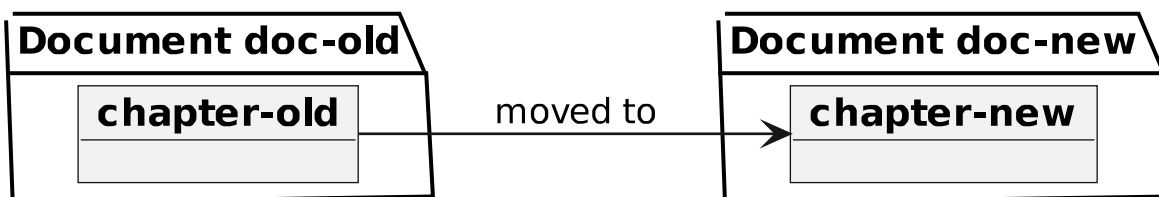
2.7 How to properly move chapter files and sections

2.7.1 Rename chapter file in same document directory



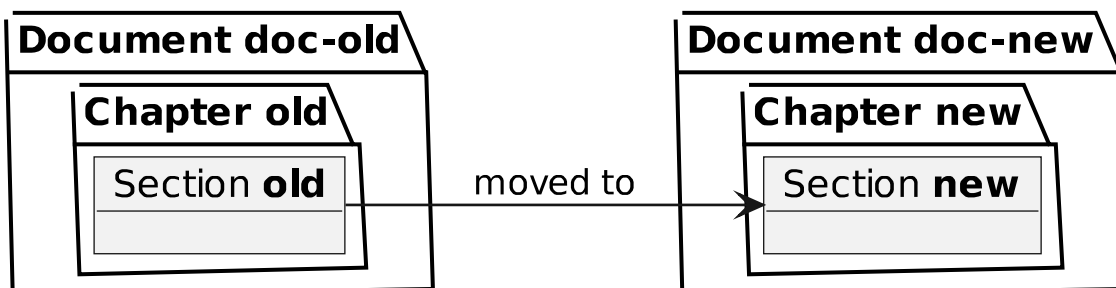
- Move `README-chapter-old.txt` to `README-chapter-new.txt`.
- Execute `M-x grep-find RET chapter-old RET`.
Rename info for **chapter-old** to info for **chapter-new**.
- Clean + compile **doc-new**.
Analyze error messages and correct errors.

2.7.2 Move chapter file to other document directory



- Move `README-chapter-old.txt` from **doc-old** to `README-chapter-new.txt` in **doc-new**.
- Execute `M-x grep-find RET chapter-old RET` in **doc-old**.
Keep document references from `doc/doc_def_standalone.inc` in `doc/doc_def.inc`.
Move and rename info for **chapter-old** from **doc-old** to **doc-new**.
- Clean + compile **doc-new**.
Analyze error messages and correct errors.
E.g. get missing substitutions/references/images/data from **doc-old**.
- Clean + compile **doc-old**.
Analyze error messages and correct errors.

2.7.3 How to move a section from one chapter file to another chapter file



- Remove **section-old** section from **chapter-old**.

Insert as **section-new** in **chapter-new**.

- Clean + compile **doc-new**.
Analyze error messages and correct errors.
E.g. get missing substitutions/references/images/data from **doc-old**.
- Clean + compile **doc-old**.
Analyze error messages and correct errors.

2.7.4 Traceability of section modifications

!todo: incorrect, needs to be updated

Case-by-case analysis:

1. Move sections before chapter documents:
 - Section **section-old** moved from **chapter-old** to **chapter-new** in **doc-old**
 - **chapter-new** moved from **doc-old** to **chapter-new** in **doc-new**
 - **chapter-new** moved to **chapter-new-new** in **doc-new**

Consequences:

- **doc-old** no longer exists. It must be looked up in the document processing logs.
- **doc-new** no longer exists. It must be looked up in the document processing logs.
- The version history of the section modification resides in the old repositories (**doc-old**, :doc-new).

2. Move chapter documents before sections:

- **doc-old** moved to **doc-old-new**
- **doc-new** moved to **doc-new-new**
- Section **section** moved from **doc-old-new** to **doc-new-new**

Consequences:

- The version history of the section modification resides in the new repositories (**doc-old-new**, :doc-new-new), which is better.

2.8 ReST section overlines

manual (book) class:

File	Book title	Chapter	Section	Subsection	Paragraph	Subparagraph
README.txt	###	===	---	~~~	...	
README-chapter-topic.txt		###	===	---	~~~	...

howto (article) class:

File	Article title	Section	Subsection	Paragraph	Subparagraph
README.txt	###	===	---	~~~	...
README-chapter-topic.txt	###	===	---	~~~	...

SDA CHAPTER NEW IS FAULTY

The help of `sda chapter --help` states:

```
COMMAND CHAPTER
chapter [SUB-COMMAND] [COMMON-OPTIONS] [ARGS]

COMMON-OPTIONS
| option          | args | description          |
| -p, --preserve-case |      | do not lowercase title for filename suffix |
| -h, --help      |      | show help           |

chapter new CHAPTER-FILENAME-OR-TOPIC
* Convert CHAPTER-TOPIC to a canonical name CANONICAL-CHAPTER-TOPIC
  which is suitable as filename README-CANONICAL-CHAPTER-TOPIC.txt

* Generate README-CANONICAL-CHAPTER-TOPIC.txt, if it does not exist.
```

However, this is not how the command actually works.

With an existing file `README-Check-Chapter-New.txt`, the command

```
sda chapter new README-Check-Chapter-New.txt
```

takes no action. No canonicalization whatsoever is performed.

3.1 From Gibberish to Brilliant Clarity

It is obvious, that the phrasing of the help text is less than clear and should be more precise to produce a meaningful command description.

3.1.1 BNF Is a Tool That Needs To Be Mastered As Such

It always helps to construct a BNF syntax declaration or at least have it in mind to clearly identify the parts of a syntactic entity.

A BNF syntax is constructed top-down. It starts with top-level syntax elements, which are then further defined until only terminal definitions or transformations are used.

The production of a BNF syntax is extremely simple:

- define top-level element with terminal and non-terminal elements
- for each undefined element, append a definition placeholder to the end of the syntax description
- define the next placeholder element

Starting with the input *CHAPTER-FILENAME-or-TOPIC*, the BNF syntax is as follows.

```
// input
CHAPTER-FILENAME-or-TOPIC:
    CHAPTER-FILENAME | TOPIC

CHAPTER-FILENAME:
    PREFIX TOPIC-SUFFIX EXTENSION

TOPIC:
    "Mixed Case Topic with Spaces"

PREFIX:
    "README"

TOPIC-SUFFIX:
    "-" FN-TOPIC | /* empty */

EXTENSION:
    ".txt"

FN-TOPIC:
    subst("[^-0-9A-Za-z], "-", TOPIC)
```

The output of the operation requires a *CANONICAL-CHAPTER-FILENAME*, which is constructed from some fixed string elements and a transformed *canonical-topic* derived from a *TOPIC* which is one possible input. The other possible input is a *CHAPTER-FILENAME*, which contains a *TOPIC* and must therefore be deconstructed.

```
// output
CANONICAL-CHAPTER-FILENAME:
    PREFIX CANONICAL-TOPIC-SUFFIX EXTENSION

CANONICAL-TOPIC-SUFFIX:
    "-" FN-CANONICAL-TOPIC | /* empty */

FN-CANONICAL-TOPIC:
    subst("[^-0-9A-Za-z], "-", canonical-topic)

canonical-topic:
    lowercase(TOPIC)
```

It is more appropriate to describe the simpler transformation of a generic *TOPIC* into a *canonical-topic* and a *CANONICAL-TOPIC-SUFFIX* before describing the deconstruction of a *CHAPTER-FILENAME* and then again describing the transformation of the generic *TOPIC* into a *CANONICAL-TOPIC-SUFFIX*.

The requirement of a *CHAPTER-FILENAME* as input is too strict, it is better to accept the name of any existing file, construct a proper chapter filename and rename the input file (*FILE-NAME*).

With option *-preserve-case* the chapter filename output is not canonical.

All of this should be reflected in the underlying BNF syntax tree:

```
// input
TOPIC-or-FILENAME:
    TOPIC | FILENAME

TOPIC:
    "Mixed Case Topic with Spaces"

FILENAME:
    FILEBASE EXTENSION | FILEBASE

FILEBASE:
    PREFIX TOPIC-SUFFIX | TOPIC

EXTENSION:
    ".txt"

PREFIX:
```

(continues on next page)

```

"README"

TOPIC-SUFFIX:
  "-" TOPIC | /* empty */

// output

CHAPTER-FILENAME:
  PREFIX TOPIC-SUFFIX EXTENSION

TOPIC-SUFFIX:
  "-" FN-TOPIC | /* empty */

FN-TOPIC:
  subst([^-0-9A-Za-z], "-", CHAPTER-TOPIC)

CHAPTER-TOPIC:
  ifelse(--preserve-case, TOPIC, canonical-topic)

canonical-topic:
  lowercase(TOPIC)

```

The description should then speak of

- *TOPIC-or-FILENAME*, deconstructed into *TOPIC* and *FILENAME*
- *canonical-topic* (to emphasize the all lowercase requirement of a canonical topic suffix)
- *CANONICAL-CHAPTER-FILENAME*

```

chapter new TOPIC-or-FILENAME
  1. If TOPIC-or-FILENAME is the name of an existing file
     - extract TOPIC from FILENAME
  2. If --preserve-case is given,
     - use TOPIC as CHAPTER-TOPIC, otherwise
     - use TOPIC converted to lowercase as CHAPTER-TOPIC.
  3. Construct FN-TOPIC by replacing all non-alpha characters
     in CHAPTER-TOPIC with dashes "-" and squeezing multiple
     dashes into a single dash.
  4. Construct CHAPTER-FILENAME from FN-TOPIC as:
     "README" "-" FN-TOPIC ".txt"
  5. If CHAPTER-FILENAME exists
     - if FILENAME exists
       - if FILENAME is different from CHAPTER-FILENAME
         - warn about existing CHAPTER-FILENAME for FILENAME
       - warn about existing CHAPTER-FILENAME for TOPIC
     - if FILENAME exists
       - if FILENAME is different from CHAPTER-FILENAME
         - rename FILENAME as CHAPTER-FILENAME
     - Create new CHAPTER-FILENAME with title TOPIC.

```

```
"README-" canonical-chapter-topic ".txt"
```

FIGURES

For best results, figures, tables and code blocks should get captions and they should be referenced with and `:xref:` for figures/tables/code-blocks (`:sref:` for sections). `:xref:` and `:sref:` are defined as `:numref:`, when appropriate, otherwise as `:ref:`. Do not use the `image` directive to keep figures in place.

Label prefixes allow using the same caption for different entities, e.g., listing and figure (see [section 4.1.1, Directive figure](#)).

Emacs support functions for generating labels and their shortcuts are shown in [table 4.1](#).

table 4.1: Emacs support functions for generating labels

shortcut	command
C-c r l s	M-x sdx-make-section-label
C-c r l f	M-x sdx-make-figure-label
C-c r l t	M-x sdx-make-table-label
C-c r l l	M-x sdx-make-listing-label
C-c r l c	M-x sdx-make-caption-label
C-c r l r	M-x sdx-make-figctr-label

4.1 Figures with numbers

Sphinx allows certain entities to be referenced by number (see [figure 4.1](#)).

```
See :xref:`fig:UML test` in :sref:`sec:Directive uml`.
```

renders as:

See figure 4.3 in section 4.1.4, Directive *uml*.

figure 4.1: Numeric references example

This requires the option `numfig` to be set to `True` in `conf.py` as shown in [listing 4.1](#).

listing 4.1: option `numfig` in `conf.py`

```
1 numfig = True
2 numfig_format = {
3     'section': 'section {number}, {name}',
4     'figure': 'figure %s',
5     'table': 'table %s',
6     'code-block': 'listing %s',
7 }
```

The supported entities are:

- **section** labels.
- Directive **figure** with filename as arguments, first paragraph in body as title, other paragraphs as legend (see [section 4.1.1, Directive figure](#)).

- Directive **table** with title (caption) as arguments, body must contain a table (see section 4.1.2, Directive *table*).
- Directive **code-block** with option `:caption:` (see section 4.1.3, Directive *code-block*).
- Directive **uml** with option `:caption:` in *figure* namespace (see section 4.1.4, Directive *uml*).
- Directives **graphviz**, **dot** with option `:caption:` in *figure* namespace (see section 4.1.5, Directives *graphviz*, *dot*).
- Directive **figctr** with title (caption) as arguments in *figure* namespace. Body contains arbitrary material (see section 4.1.6, Directive *figctr*).

4.1.1 Directive *figure*

As documented in reStructuredText Directives, *figure*. See listing 4.2) and figure 4.2).

- **always** use **fig:** prefix for label

listing 4.2: Figure example Magritte's Pipe

```
1 .. _`fig:Figure example Magrittes Pipe` :  
2 .. figure:: _static/MagrittePipe.jpg  
3  
4 Figure example Magritte's Pipe
```



figure 4.2: Figure example Magritte's Pipe

4.1.2 Directive *table*

The body must contain a table, however, a single table cell (rectangle) is sufficient (see listing 4.3 and table 4.2).

- **always** use **tab:** prefix for label

listing 4.3: table directive with single table cell (rectangle)

```

1 .. _`tab:single table cell rectangle`:
2 .. table:: single table cell (rectangle)
3
4 +-----+
5 | .. code-block:: text |
6 | |
7 |     code-block without |
8 |     caption inside a  |
9 |     single table cell |
10 +-----+

```

table 4.2: single table cell (rectangle)

```

code-block without
caption inside a
single table cell

```

4.1.3 Directive *code-block*

code-block directives with a *:caption:* option can be referenced by number, e.g., “*See :xref:‘lst:Definition of a general code-block listing’*”: renders as “See listing 4.4”.

- **always** use **lst:** prefix for label
- **always** specify language
- **always** use option *:linenos:*

listing 4.4: Definition of a general code-block listing

```

1 .. _`lst:<LABEL>`:
2 .. code-block:: <LANGUAGE>
3   :caption: <TITLE>
4   :linenos:
5
6   <BODY>

```

An example is shown in listing 4.5.

listing 4.5: Example with Elisp code

```

1 .. _`lst:Some Emacs Lisp code`:
2 .. code-block:: elisp
3   :caption: Some Emacs Lisp code
4   :linenos:
5
6   (if (not nil)
7       (let (x) (princ x))
8       (let (y) princ y))

```

The rendered example is shown in listing 4.6.

listing 4.6: Some Emacs Lisp code

```

1 (if (not nil)
2   (let (x) (princ x))
3   (let (y) (princ y)))

```

4.1.4 Directive *uml*

The *uml* directive for plantuml(1) diagrams also works as figure, as shown in listing 4.7 and figure 4.3.

- always use **fig:** prefix for label

listing 4.7: uml directive for plantuml(1) diagrams

```

1 .. _`fig:UML test`:
2 .. uml::
3   :caption: UML test
4
5   @startuml
6   A --> B : send
7   A <-- B : receive
8   @enduml

```

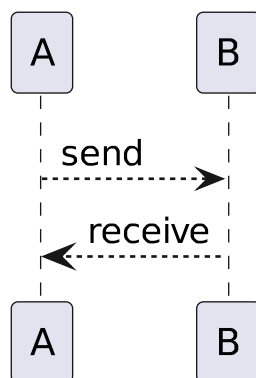


figure 4.3: UML test

4.1.5 Directives *graphviz*, *dot*

Obviously, the *graphviz* directive for dot(1) diagrams also works as figure, as shown in listing 4.8 and figure 4.4.

- always use **fig:** prefix for label

listing 4.8: *graphviz* directive for dot(1) graph diagrams

```

1 .. _`fig:dot test`:
2 .. graphviz::
3   :caption: dot test
4
5   digraph xx {
6     A -> B;
7   }

```

4.1.6 Directive *figctr*

Since single cell tables end up in the table namespace, the extension *ws_figure_container* provides the *figctr* directive, which works like a container in the figure namespace (see listing 4.9 and figure 4.5).

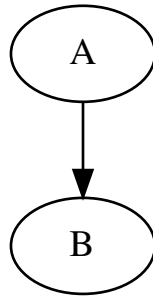


figure 4.4: dot test

listing 4.9: figctr directive

```

.. _`fig:Fig Ctr Caption`:
.. figctr:: Fig Ctr Caption

Arbitrary things like this:

.. code-block:: sh

    if test -r "file"
    then
        cat "file"
    fi

```

Arbitrary things like this:

```

if test -r "file"
then
    cat "file"
fi

```

figure 4.5: Fig Ctr Caption

4.2 See also

- [How to number figures in Sphinx. How to do subfigures](#)

4.3 Check README for figure requirements

Before a README is done, it is useful to check, if every figure has a caption, directive and a reference.

figure-regexp:

```

^ *\[([.][.]) *\[([table|figure|uml|graphviz|graph|digraph]):\|_
↪ \[([tab|fig]):\|:caption:\|:xref:
figure-grep-cmd::

```

(continues on next page)

(continued from previous page)

```

/bin/grep --color -nH README-fillme.txt -e '{figure-regexp}'

Search with

- :kbd:`M-x occur RET {figure-regexp} RET`
- :kbd:`M-x grep RET {figure-grep-cmd} RET`

- For creating figure labels, use emacs commands

  - :kbd:`C-c r l c`, :kbd:`M-x sdx-make-caption-label`
  - :kbd:`C-c r l f`, :kbd:`M-x sdx-make-figure-label`
  - :kbd:`C-c r l t`, :kbd:`M-x sdx-make-table-label`

See :xref:`fig:complete figure requirements`

.. _`fig:complete figure requirements`:
.. uml::
   :caption: complete figure requirements

   @startuml
   package "complete figure requirements" {
     start
     floating note right
     figure requirements:
     * caption
     * directive
     * reference
     end note
     :grep for uml/table (figure-regexp);
     while (for each matched directive line) is (do)
       while (caption, label or reference is missing?)
         if (caption is missing?) then (yes)
           :* add caption
           * correct label, if present
           * correct reference, if present;
         elseif (label is missing?) then (yes)
           :* move before figure
           * add label with `C-c r l [cft]`
           * correct reference, if present;
         else (reference\nis missing)
           :add reference;
         endif
       end while
     end while
     stop
   }
   @enduml

```

4.4 Templating with automatic labels (NO!)

:q:| :da:| f5 Sections (3, 4, 5) yank a new section title, why not also a .. sec:fillme: above it?
:a:| because:

- Do not inflate the use of section labels! if a section is **not** referenced, it should **not** have a label.
- Why would want write the section title three times?

The correct template would be:

```

:xref:`sec::fillme:`
.. _`sec::fillme:`

=====
:sec: ::fillme::
=====

```

Filling in the section title must be done anyway:

- change tag delimiter to “:.” (C-c C-x)
- move to beginning of *fillme* (M-left)
- delete *fillme* (M-k)

Alternative 1: Copy title and replace label *fillme*

- mark beginning of title (C-SPC)
- enter title
- copy title (M-w)
- go to label *fillme* (M-left)
- delete *fillme* (M-k)
- insert title (C-y)
- Clean up label (remove all non-ascii characters) 0 - aleph_0 keystrokes
- If the title ws not clean, copy the label title (many keystrokes)
- go to :xref: *sec:::fillme::* (M-left)
- delete *fillme* (M-k)
- insert title (C-y)

Minimum keystrokes: 8, Maximum keystrokes: aleph_0

Alternative 2: Write title and generate label with emacs function (C-c r l s)

- enter title
- go before :sec: line (up)
- generate xref + label (C-c r l s)

Minimum keystrokes: 5, Maximum keystrokes: 5

CITATIONS

References are collected in `doc/bibliography.inc`, which contains some elisp helpers for citing RFCs and Wikipedia.

Note: Citations are realized as regular labels/hyperlinks. So ``WPABBR`_` and `[WPABBR]_` reference the same target, if they are defined in the same document. `[WPABBR]_` also works, if the definition is in another document, while ``WPABBR`_` is undefined in that case.

GLOSSARY

Glossary and abbreviations are generated by `sphinx_doc_glossary.py` from the common source `doc/glossary.src`.

6.1 As for the style of glossary entries

Either use dictionary/encyclopedia style[`DICTTERM`]

term a word or group of words designating something, especially in a particular field, as atom in physics, quietism in theology, adze in carpentry, or district leader in politics.

or section style (term assumes the role of a section title)

Term Words, compound words or multi-word expressions[`WPTERM`]

[...] that in specific contexts are given specific meanings - these may deviate from the meanings the same words have in other contexts and in everyday language.

Do not mix styles.

Do not write a glossary entry as a single sentence:

This example is very very bad. **Never ever** write like this.

Have a look at the google search [glossary entries example](#).

6.2 Multiple *glossary* directives

It is possible to have multiple glossaries spread throughout a document. This is fine for hypermedia (HTML, PDF, EPUB). However, if the document is to be printed it is a very bad idea to have glossaries other than *abbreviations* and *glossary*, since a reader will not be able to locate them easily.

Both, the list of abbreviations, as well as the glossary are each defined with a *glossary* directive:

Source ReST markup	Rendered output
<pre> Abbreviations ===== .. glossary:: abbr a black blade runner first see :term:`f i r s t` and short :term:`abbr` </pre>	<p>Abbreviations</p> <p><i>abbr</i> a black blade runner first see <i>f i r s t</i> and short <i>abbr</i></p>
<pre> Glossary ===== .. glossary:: abbr a black blade runner some description a black blade runner elaborate entry for abbr f i r s t see :term:`first` </pre>	<p>Glossary</p> <p>abbr a black blade runner some description a black blade runner elaborate entry for <i>abbr</i> f i r s t see <i>first</i></p>
<pre> .. abbr replace:: :term:`abbr` <a black_ ↪blade runner>` </pre>	

6.2.1 Order of abbreviations and glossary

In case of duplicate glossary terms in multiple glossaries, the last entry wins when resolving a reference to the term. Therefore, abbreviations should appear before the glossary. Here is an exact term reference *abbr* (:term:`abbr`) and an elaborate one *abbr* (.. |abbr| replace:: :term:`tstaddr` <a black blade runner>`).

6.3 Combinations of glossary and abbreviation generation

Enable features with option `-enable FLG,FLG,...`:

gl[glossary] option: `glo_enabled` generate `glossary.inc`

gt[erm] option: `glo_term_enabled` add term+definition to glossary

ga[bbrev] option: `glo_abbr_enabled` add abbr+term/definition to glossary

ab[brevs] option: `abbr_enabled` generate `abbrevs.inc`

ad[efs] option: `adef_enabled` generate `abbrev_defs.inc`

f[glossary] option: `force_glossary` force abbreviation only entries as :todo: glossary entries

figure 6.1

Dflt	Glo Term?	Glo Abbr?	Glossary Entries	Abbreviation Entry	Abbreviation Subst
	+	+	abbr see :term:`a black blade runner` a black blade runner some description	abbr :term:`a black blade runner`	.. labbrl replace:: :term:`abbr` <a black blade runner>`
*	+	-	a black blade runner some description	abbr :term:`a black blade runner`	.. labbrl replace:: :term:`abbr` <a black blade runner>`
	-	+	abbr a black blade runner Some description	labbrl a black blade runner	.. labbrl replace:: :term:`abbr`
	-	-		abbr a black blade runner	.. labbrl replace:: :term:`abbr`

Column	Description
Dflt	* = default setting for sphinx_doc_glossary.py
Glo Term?	+ => glossary contains expanded term entry - => glossary does not contain expanded term entry
Glo Abbr?	+ => glossary contains abbreviation entry - => glossary does not contains abbreviation entry
Glossary Entries	Entries generated for glossary
Abbreviation Entry	Entry generated for list of abbreviations
Abbreviation Subst	Subst definition for abbreviation

figure 6.1: Combinations of Glossary and Abbreviation Generation

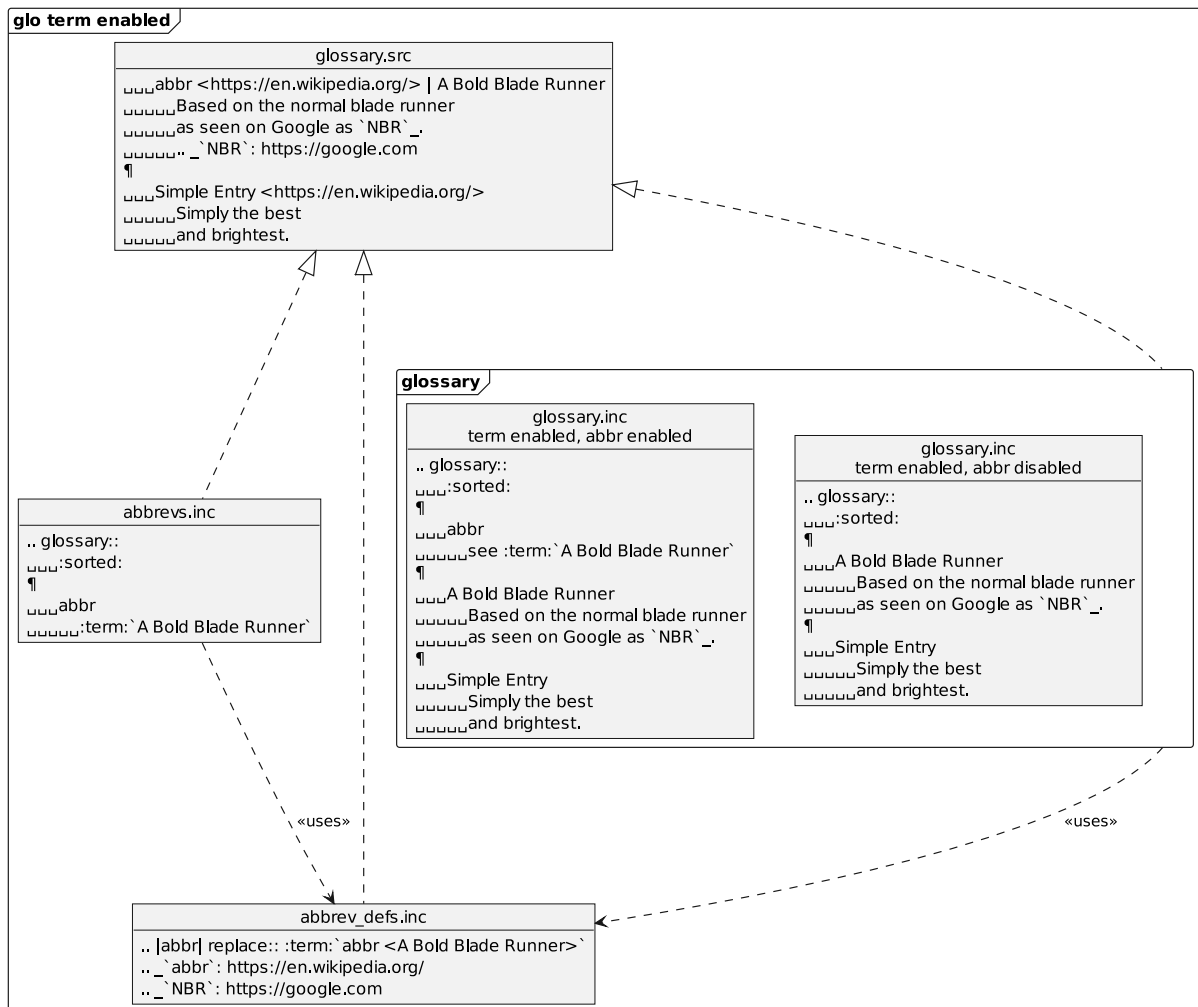


figure 6.2: Generated output files for glossary terms enabled

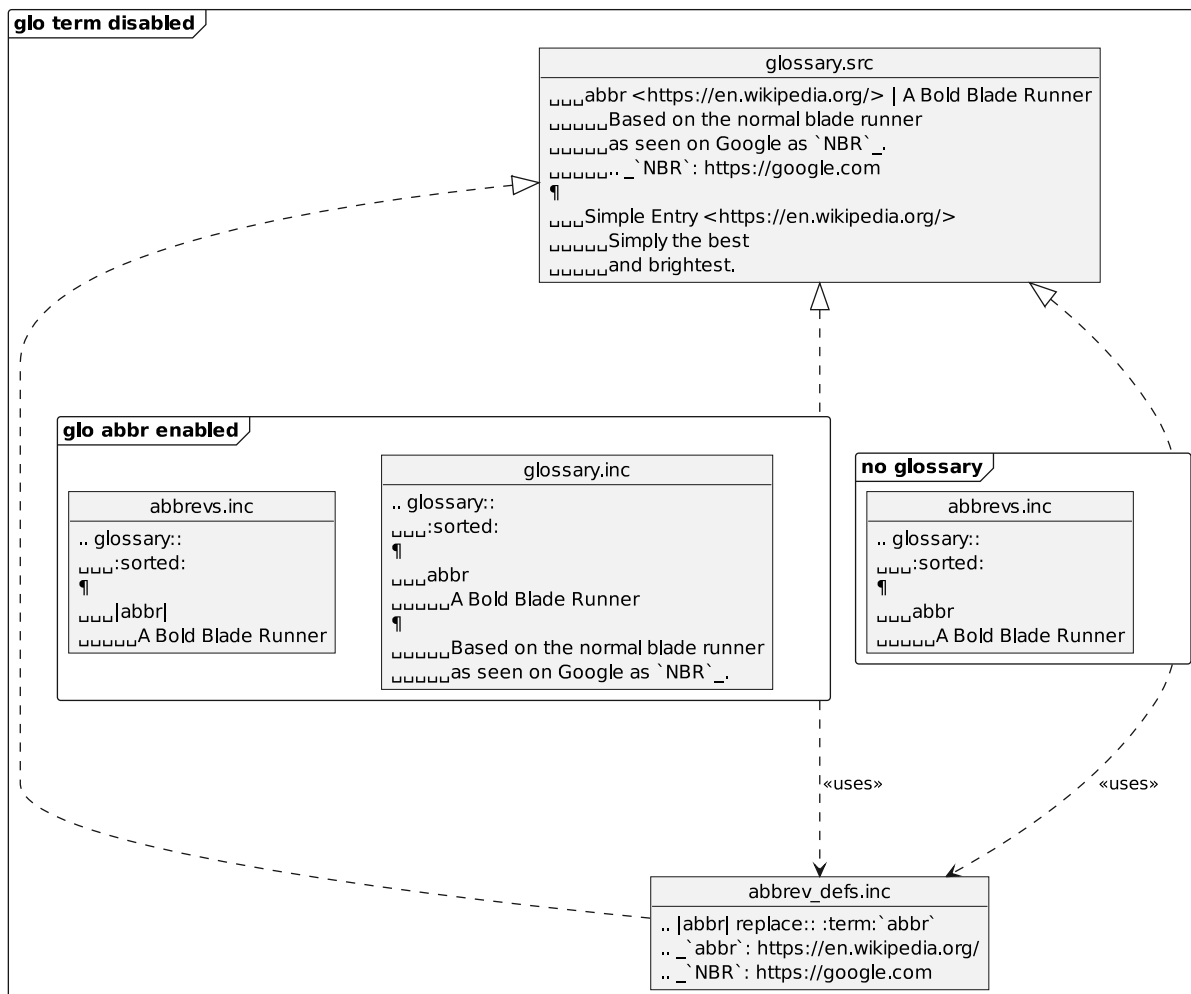


figure 6.3: Generated output files for glossary terms disabled

6.3.1 Object diagrams

6.3.2 Class diagram



figure 6.4: Glossary class

6.3.3 Activity diagrams

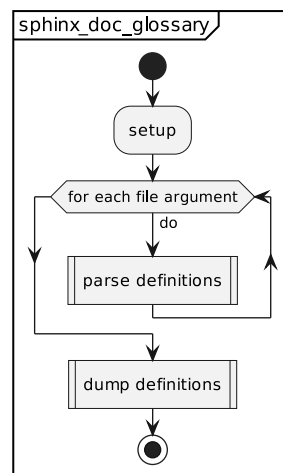


figure 6.5: Glossary parser activity diagram

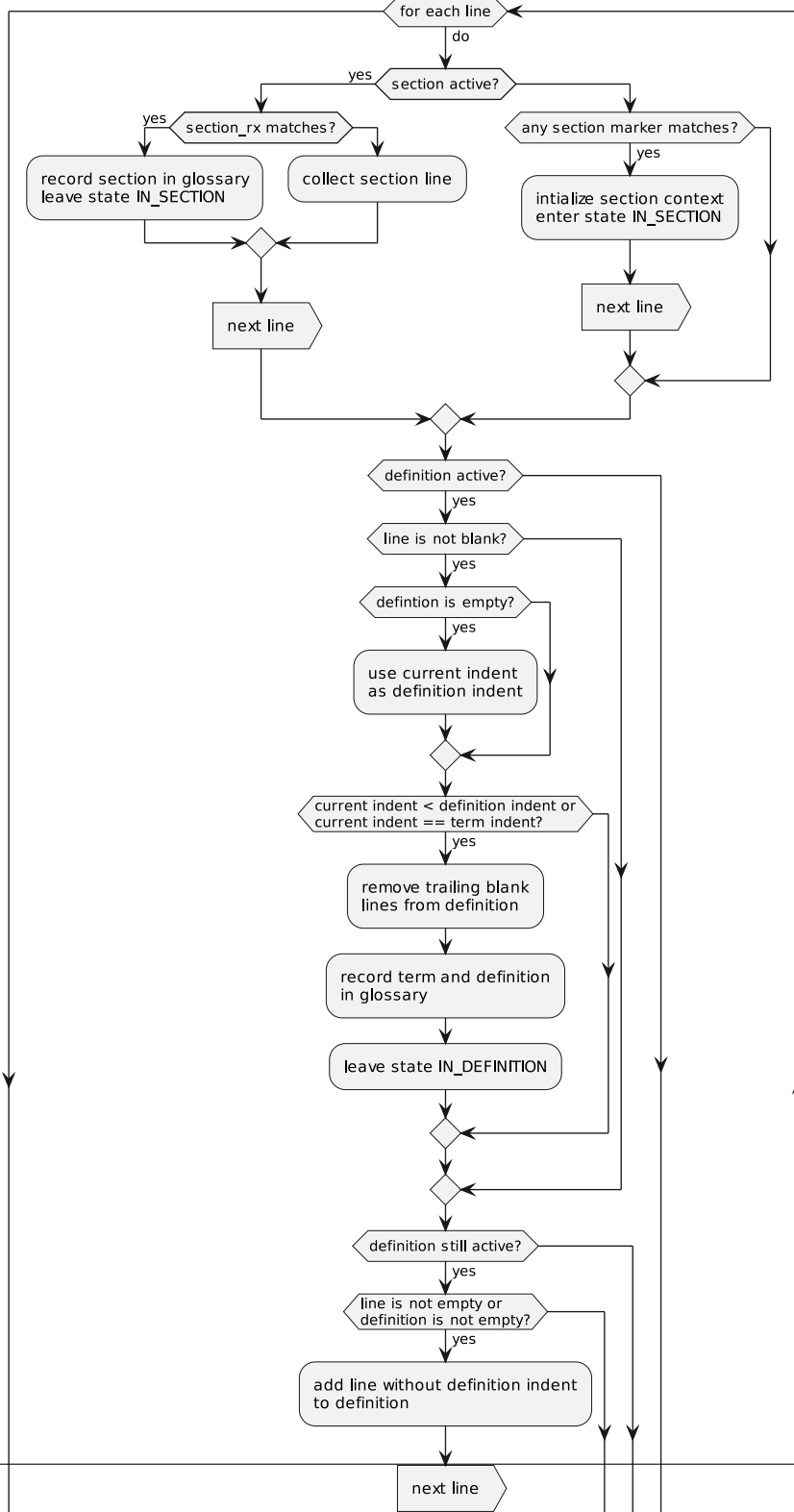
6.3.4 State diagram

parse glossary definitions

parse term/definition block:
 [abbreviation [<abbrev link>]] term [<term link>]
definition line
definition line
 [subst] replace:: **subst**
 _link: http://some.where.com

Blank lines between term and first non-empty line of definition are ignored.

link and subst definitions are diverted to *abbrev_defs.inc*



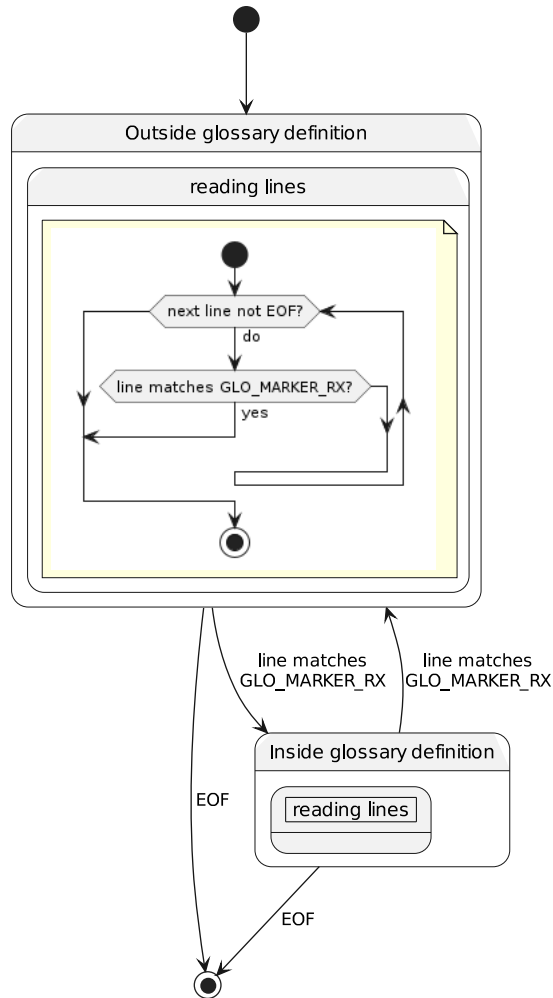


figure 6.7: Glossary parser state diagram

UML ANNOTATIONS - LINE_DIVERSION

The python module `line_diversion` extracts marked annotation lines from text files. The extraction result is generally a PlantUML diagram. Future extensions to extract other material (e.g. `dot(1)` graphs) are possible and probable.

While writing source code, the PlantUML annotations can be easily added near the source code lines. (See section *Emacs support*). The idea of this concept is to minimize the distance between source code and documentation (see section 8, *Relevance of Documentation*)¹.

7.1 Emacs support

A set of Emacs commands facilitates code annotation, reformatting of annotations and diagram preview. The commands are all prefixed with `umlx-` and bound with the prefix command `C-# u`. See `C-# u C-h` for key bindings:

<code>C-# u a</code>	<code>umlx-mark-activity</code>
<code>C-# u b</code>	<code>umlx-re-mark-buffer</code>
<code>C-# u c</code>	<code>umlx-convert-old</code>
<code>C-# u g</code>	<code>umlx-grep-find</code>
<code>C-# u k</code>	<code>umlx-unmark</code>
<code>C-# u m</code>	<code>umlx-mark</code>
<code>C-# u o</code>	<code>umlx-occur</code>
<code>C-# u t</code>	<code>umlx-symbol-tag-delimiter</code>
<code>C-# u u</code>	<code>umlx-re-mark</code>
<code>C-# u x</code>	<code>umlx-extract</code>

In addition to the `C-# u <char>` binding, some commands are also directly bound to `C-# <char>`:

<code>C-# C-a</code>	<code>umlx-mark-activity</code>
<code>C-# C-b</code>	<code>umlx-re-mark-buffer</code>
<code>C-# C-k</code>	<code>umlx-unmark</code>
<code>C-# RET</code>	<code>umlx-mark</code>
<code>C-# C-t</code>	<code>umlx-symbol-tag-delimiter</code>
<code>C-# C-u</code>	<code>umlx-re-mark</code>

Note: If `C-#` does not work (e.g. in a terminal), the prefix command `C-c #` can be used instead.

7.2 Annotation tags and markers

An annotation marker consists of

- an annotation tag, which consists of

¹ Normally, PlantUML code is placed before the source code of a script, class, function etc. or in a separate text file. In this case it is necessary to have two windows, one for documenting the source code as PlantUML and the other for coding. This can lead to the phenomenon of big differences between the source code and its documentation, because of laziness. To prevent this from happening, it is useful to use UML annotations.

- a comment start sequence,
- followed directly (without whitespace) by a tag symbol, which consists of
 - * a `line_diversion` type
 - * and a diagram number,
- optional type parameters,
- optional text
- and an optional comment end sequence.

ANNOTATION-MARKER:

```
<ANNOTATION-TAG> [<SPACE> TYPE-PARAMETERS ..] [<SPACE> <TAIL-TEXT>] [[<SPACE>] <COMMENT-END>]
```

ANNOTATION-TAG:

```
<COMMENT-START> <ANNOTATION-TAG-SYMBOL>
```

ANNOTATION-TAG-SYMBOL:

```
<LINE_DIVERSION-TYPE> <DIAGRAM-NUMBER>
```

E.g.:

```
#a55
a #a55 :;
b #a55 :; #red
#a55 ' (yes)
```

An annotated line is defined as:

```
[[<SPACE>] <COMMENT-START> <SPACE> <TEXT>] <ANNOTATION-MARKER>
| [[<SPACE>] <KEYWORD> <ANNOTATION-MARKER>
| <TEXT> <ANNOTATION-MARKER> <SPACE> : [; ]
```

7.2.1 Comment start regular expression

The regular expression for matching a comment start is defined in module `line_diversion`:

```
COMMENT_TYPE_RX='(?://+|\\/+|;+|@:u?[bl]?comm_?@|--|<!--|#+|@[bl]?comm_?@[.].)'
```

7.2.2 Line diversion types

The line diversion types recognized are also defined in module `line_diversion`:

Marker	Diagram	
#c[0-9]+	class diagram	
#o[0-9]+	object diagram	
#p[0-9]+	component diagram	
#d[0-9]+	deployment diagram	
#u[0-9]+	use case diagram	
#a[0-9]+	activity diagram	
#s[0-9]+	state machine diagram	
#m[0-9]+	sequence diagram	
#t[0-9]+	timing diagram	

Please refer to `line_diversion.py --help` for authoritative information.

7.3 Practical annotation

Most, but not all annotations are comments. `line_diversion` also recognizes some annotated code lines like `class`, `if`, `fi`, ... Arbitrary code lines can be marked as activity.

The first thing to do is to define an annotation tag symbol (in the example it is for an activity diagram). For activity diagrams, use `a0`, `a1`, `a2`, ... However, the diagram numbers are strictly informational and can be arbitrary. The numbers are not interpreted as integers, i.e. `a0`, and `a00` are different annotation tag symbols.

The tagging commands `C-# m`, `C-# a`, etc. use the last specified `ANNOTATION-TAG-SYMBOL`. They will ask for the `ANNOTATION-TAG-SYMBOL` to be used, when invoked with a prefix argument, e.g. `C-u C-# m`.

Note that `line_diversion.py` displays diagrams in the order they are first encountered in the text, not sorted by diagram numbers.

Annotation lines can be distributed anywhere throughout the source text. The annotation text is the same as regular `PlantUML` commands, with a few exceptions:

- `@startuml` and `@enduml` are implicitly added by `line_diversion.py` and must not be explicitly defined,
- the prefix and postfix of **actions in activity diagrams** (e.g. `:activity ;`) can be written after the [PlantUML](#) marker.

Here is an unannotated Python example:

```
# display `hello world`
printf ('hello world')
```

So instead of adding `:`, adding `;` and marking the annotation with `C-# m`:

```
# :display `hello world`; #a0
printf ('hello world')
```

the annotation can be marked with `C-# a`:

```
# display `hello world` #a0 ;;
printf ('hello world')
```

For a quick and dirty session, code statements can be marked directly as activities:

```
printf ('hello world') #a0 ;;
```

However, that is not the correct way to document anything, since it is sufficient to read the source code in such a case. There is simply no need to generate a documentation which repeats the source code word for word. The exception is for a quick overview of unknown code, where the annotations are just temporary.

[PlantUML](#) elements other than actions in an activity diagram are not enclosed in special delimiters `: abnd ;`:

```
# start #a20
a = 5 #a20 ;;
b = a #a20 ;;
if (a==b): #a20 (yes)
  a = 12 #a20 ;;
else: #a20
  b = 15 #a20 ;;
# show values of **a** and **b** #a20 ;;
print(a)
print(b)
# endif #a20
# stop #a20
```

The extraction command:

```
line_diversion.py --match '^a20$' README-uml-annotations-line-diversion.txt
```

results in:

```
@startuml
skinparam padding 1
start
:a = 5;
:b = a;
if (a==b) then (yes)
  :a = 12;
else
  :b = 15;
:show values of **a** and **b**;
endif
stop
@enduml
```

This output can be rendered and previewed in place with `C-c u u v`, which is bound to the command `x-plantuml-preview-current-block`.

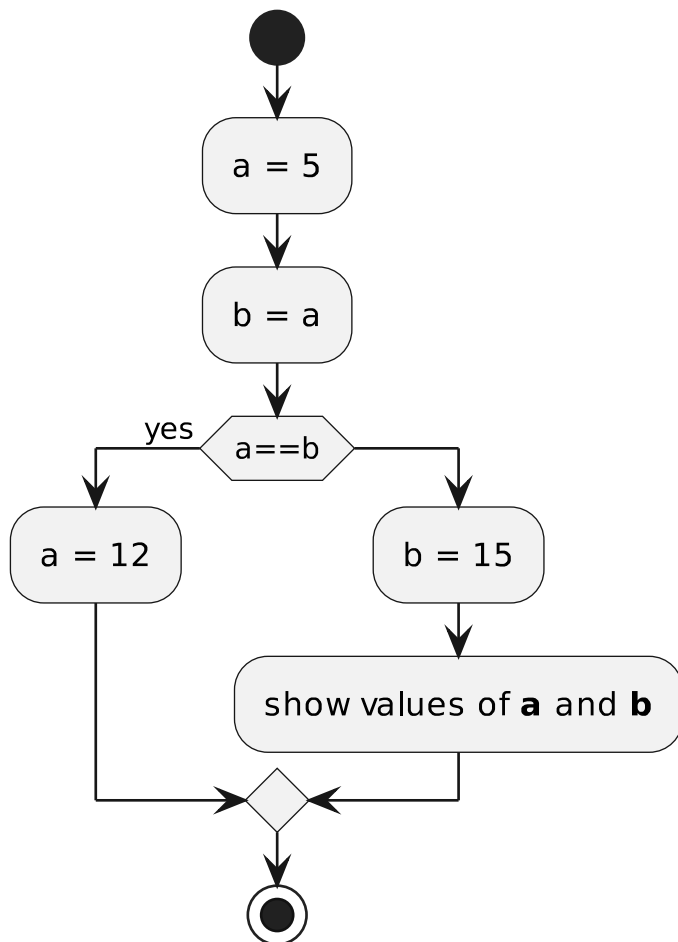
The command `umlx-extract`, which is bound to `C-# u x`, combines extraction, rendering and previewing. Without a prefix argument, the diagram for the active default marker in the current file is displayed.

With a prefix of 0, the diagram marker can be specified.

With a negative prefix argument, the diagrams are only extracted. No preview is generated.

With a positive prefix argument > 0 (and < 16), the diagram with the corresponding sequence number is rendered and displayed. As mentioned earlier, the extraction sequence number (1, 2, 3, ..) of diagrams is unrelated to the *DIAGRAM-NUMBER* of the *ANNOTATION-TAG-SYMBOL*.

The last example in the text file `README-uml-annotations-line-diversion.txt` is extracted and rendered with `C-u 0 C-# u x a20 RET` as:

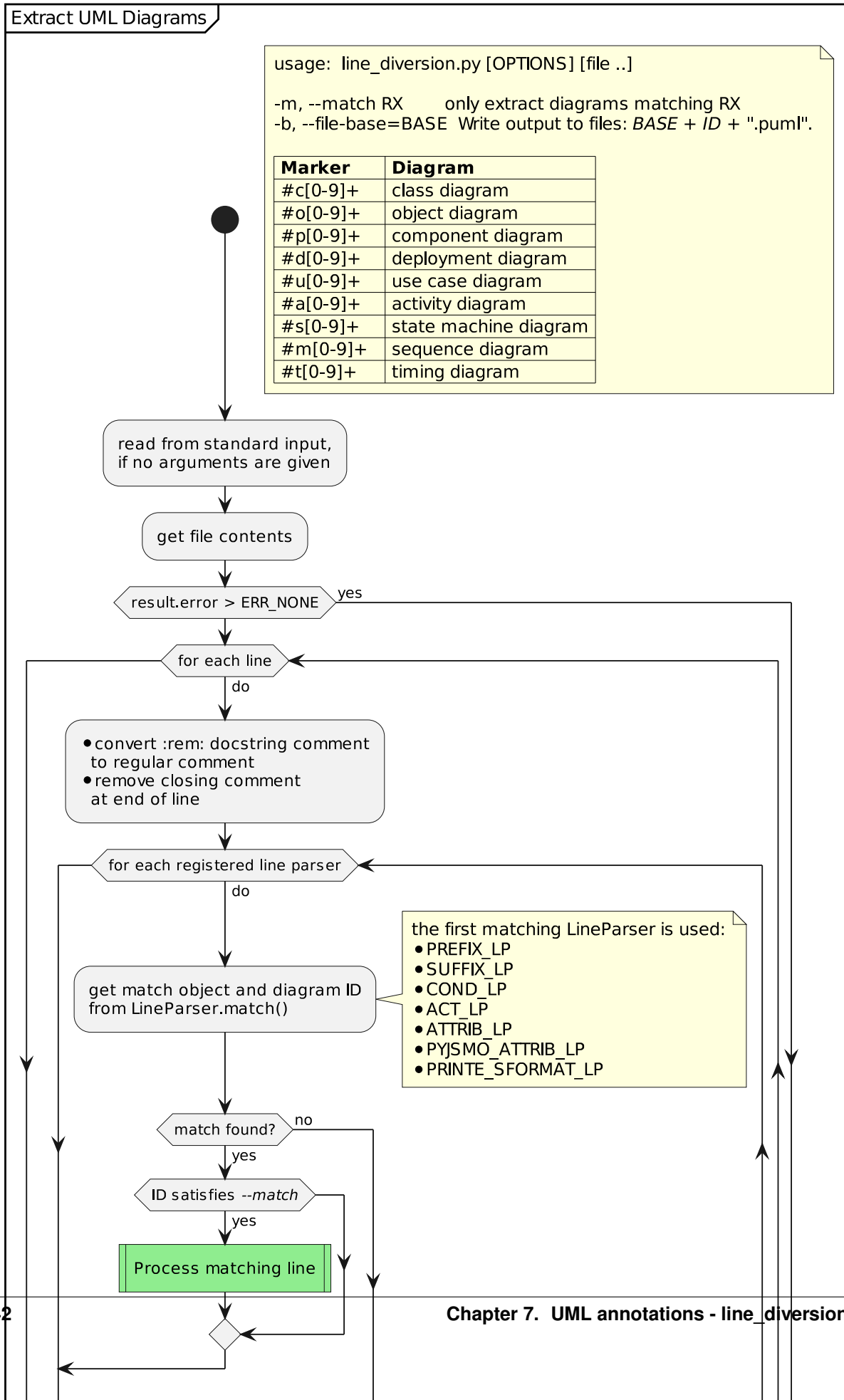


A prefix number < 0 (e.g., `C-- C-# u x` and `C-0 C-# u x`) shows the extracted [PlantUML](#) diagram definition instead of a preview. It is then very simple to walk through the output and preview diagrams with `C-c u u v`, which is bound to the command `x-plantuml-preview-current-block`. This is also convenient when debugging errors.

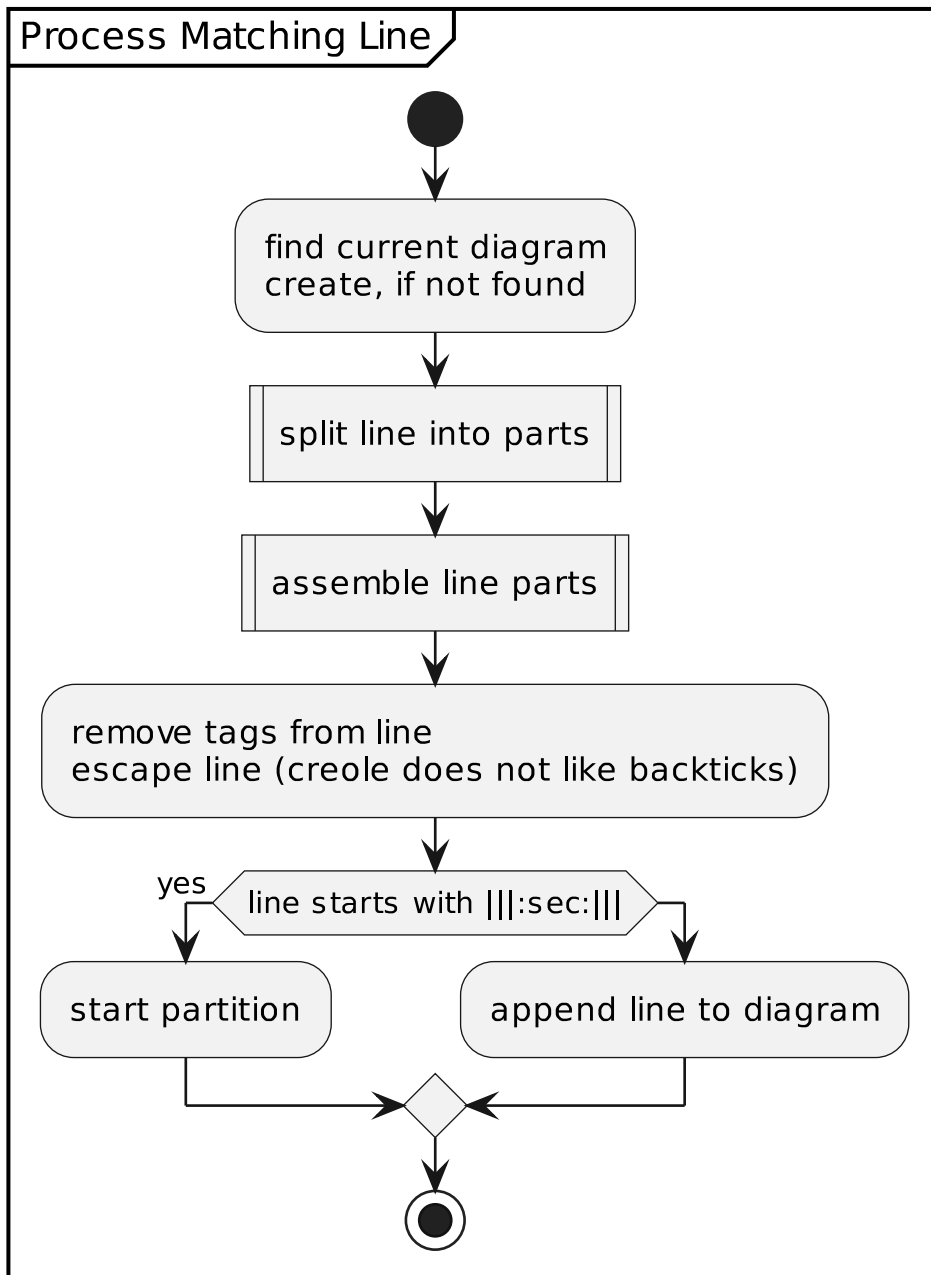
7.4 Activity Diagrams (extracted)

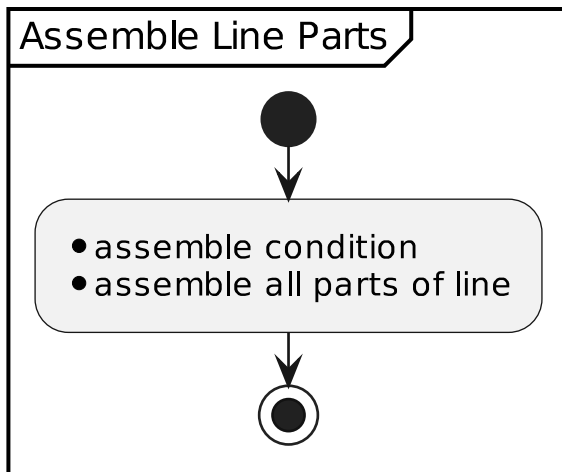
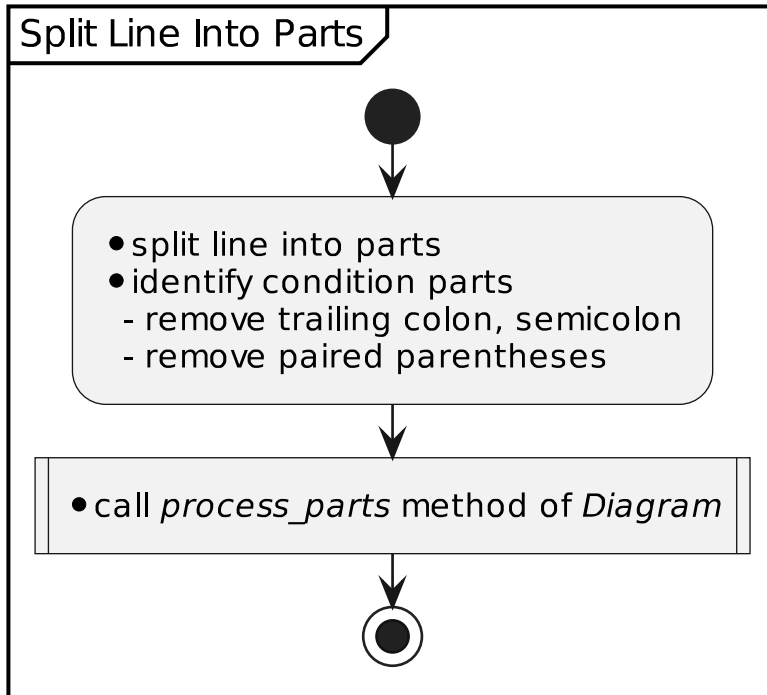
Activity diagrams for extracting UML diagrams from annotated source code:

7.4.1 Extract UML diagrams



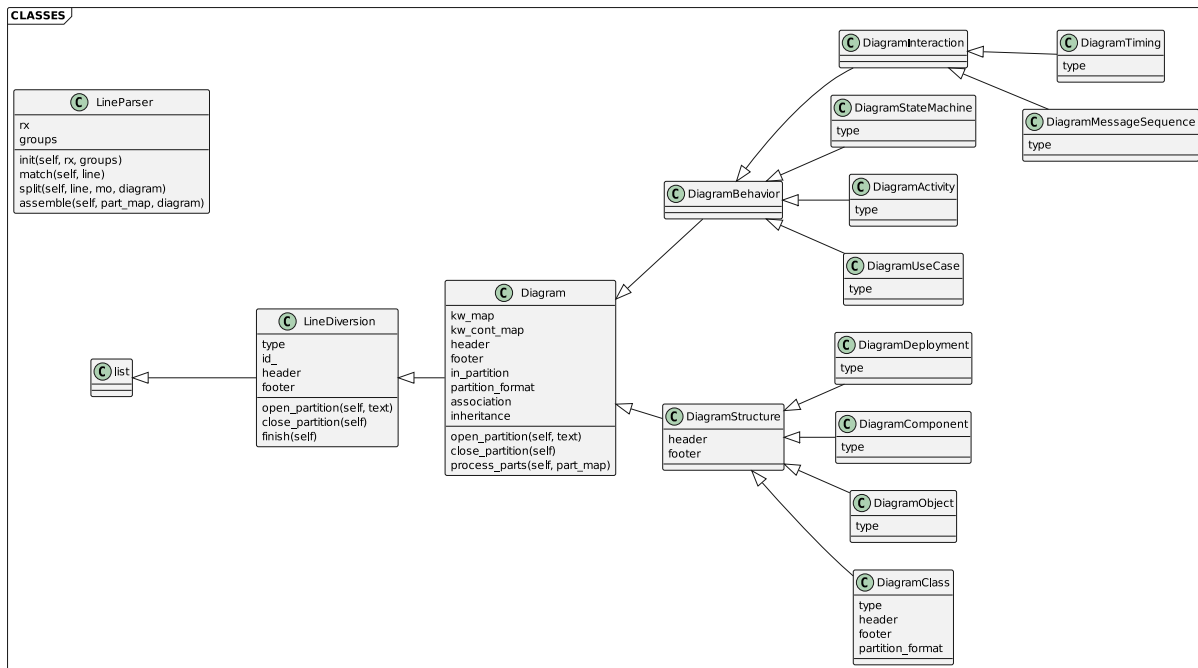
7.4.2 Process Matching Line





7.5 Class Diagram (extracted)

Class diagram:



7.6 Command/Module Documentation

line_diversion.py - extract UML diagram from source code

usage:	line_diversion.py [OPTIONS] [file ..]
or	import line_diversion

7.6.1 Options

-m, -match RX	only extract diagrams matching RX
-b, -file-base=BASE	Write output to files: <i>BASE</i> + <i>ID</i> + “.puml”. E.g. -file-base=”file-base-” => file-base-a0.puml
-show-bases	show base classes for class diagrams (default)
-hide-bases	hide base classes for class diagrams
-ignore-bases CLASS	ignore base class for class diagrams (can be specified multiple times) (default: object, PyJsMo)
-q, -quiet	suppress warnings
-v, -verbose	verbose test output
-d, -debug[=NUM]	show debug information
-h, -help	display this help message
-template list	show available templates.
-eide[=COMM]	Emacs IDE template list (implies -template list).
-template[=NAME]	extract named template to standard output. Default NAME is -.
-extract[=DIR]	extract adhoc files to directory DIR (default: .)
-explode[=DIR]	explode script with adhoc in directory DIR (default __adhoc__)
-setup[=install]	explode script into temporary directory and call <i>python setup.py install</i>
-implode	implode script with adhoc
-t, -test	run doc tests

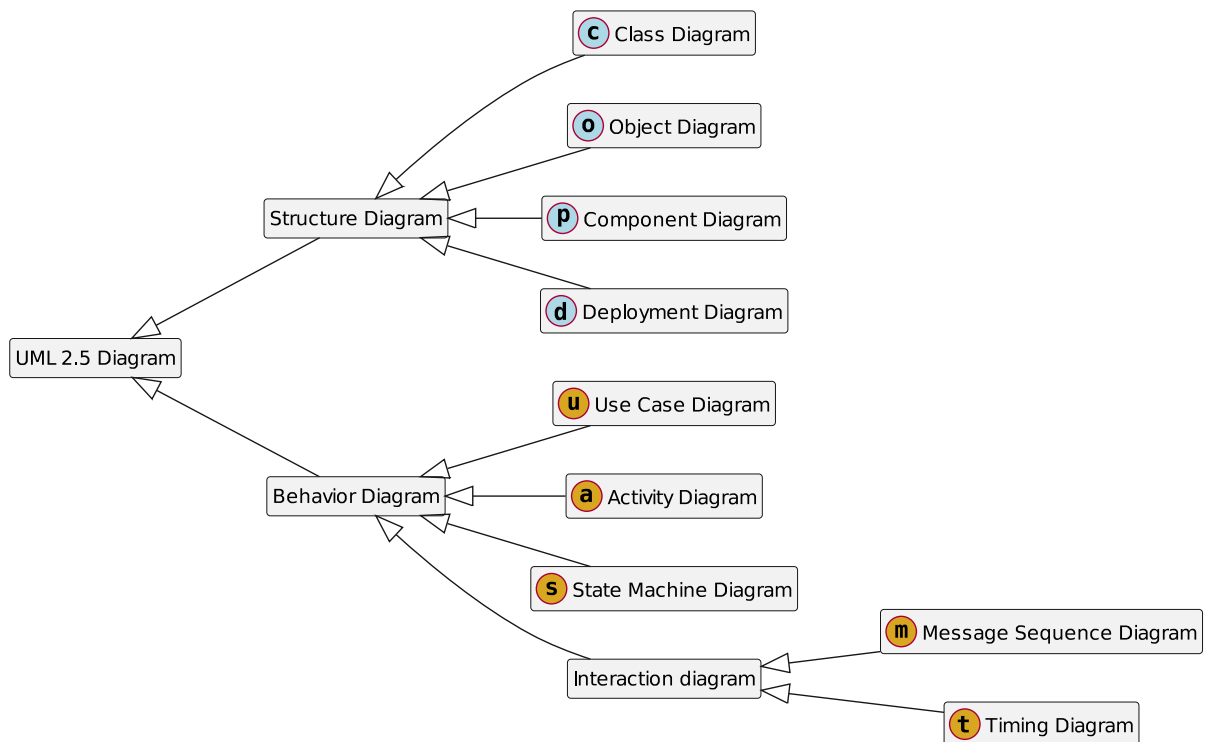
7.6.2 Module

These are the UML diagrams and type codes supported by PlantUML ..

Marker	Diagram	
#c[0-9]+	class diagram	
#o[0-9]+	object diagram	
#p[0-9]+	component diagram	
#d[0-9]+	deployment diagram	
#u[0-9]+	use case diagram	
#a[0-9]+	activity diagram	
#s[0-9]+	state machine diagram	
#m[0-9]+	sequence diagram	
#t[0-9]+	timing diagram	

• Structure Diagrams	Type	• Behavior Diagrams	Type
• Class Diagram	c	• Use Case Diagram	u
• Object Diagram	o	• Activity Diagram	a
• Component Diagram	p	• State Machine Diagram	s
• Deployment Diagram	d	• Message Sequence Diagram	m
		• Timing Diagram	t

This is the inheritance hierarchy of UML diagrams supported by PlantUML (clickable in HTML):



7.6.3 Automatic Exports

```

>>> for ex in __all__: printf(sformat('from {0} import {1}', __name__, ex))
from line_diversion import check_line_parse
from line_diversion import LineParser
from line_diversion import DiagramTiming
from line_diversion import DiagramMessageSequence
from line_diversion import DiagramInteraction
from line_diversion import DiagramStateMachine
from line_diversion import DiagramActivity
from line_diversion import DiagramUseCase

```

(continues on next page)

(continued from previous page)

```

from line_diversion import DiagramBehavior
from line_diversion import DiagramDeployment
from line_diversion import DiagramComponent
from line_diversion import DiagramObject
from line_diversion import DiagramClass
from line_diversion import DiagramStructure
from line_diversion import Diagram
from line_diversion import LineDiversion

```

7.6.4 Explicit Exports

```

>>> if '__all_internal__' in globals():
...     for ex in __all_internal__:
...         printf(sformat('from {0} import {1}', __name__, ex))

```

7.6.5 Details

See `check_line_parse()` for comprehensive list of line matchers.

Prefix Match

```

>>> printf(PREFIX_LP)
{
  "rx": [
    "^(\\s*) (?://+|/\\*+|;+|@:u?[bl]?comm_?@|--|#+|@[bl]?comm_?@[.][.]) ([copduasmt] [0-
↵9]+) (?:\\s|$)",
    "0"
  ],
  "groups": {
    "whitespace": 1,
    "id": 2,
    "text": null,
    "keyword": null,
    "condition": null
  }
}

```

```

>>> mo, _id = PREFIX_LP.match('#' 'a3 if (test) then (yes)')
>>> printf(mo.groups())
(' ', 'a3')

```

Condition Match

```

>>> printf(COND_LP)
{
  "rx": [
    "^(\\s*) (break|class|def|done|elif|else|elseif|elsif|fi|for|if|while|\\}) (?:?
↵|\\s) \\s*(.*) \\s*(?://+|/\\*+|;+|@:u?[bl]?comm_?@|--|#+|@[bl]?comm_?@[.][.]) ([copduasmt] [0-
↵9]+) (?:\\s(\\.|) |\\s*)$",
    "0"
  ],
  "groups": {
    "whitespace": 1,
    "id": 4,
    "text": 5,
    "keyword": 2,
    "condition": 3
  }
}

```

```
>>> mo, _id = COND_LP.match('          if remove_count: #'a1 then (yeah)')
>>> printf(mo.groups())
('          ', 'if', 'remove_count: ', 'a1', 'then (yeah)')
```

```
>>> mo, _id = COND_LP.match(' #'a1')
>>> printf(mo.groups())
('', ' ', ' ', 'a1', None)
```

Action Match

```
>>> printf(ACT_LP)
{
  "rx": [
    "^(\\s*)(?: (?://+|\\/+*+;+|@:u?[bl]?comm_?@|--|#+|[bl]?comm_?@[.][.]) )?(.*)\\s*(?:/
↵+|/\\/+*+;+|@:u?[bl]?comm_?@|--|#+|[bl]?comm_?@[.][.]) ([[copduasmt][0-9]+) (:[:];|<>/})?|[:];
↵|<>/}.-)]\\s*(?: ([0-9A-Za-z]+|backwards) (?:\\s+|$))?(?: \\s(.*)|\\s*)$",
    "0"
  ],
  "groups": {
    "whitespace": 1,
    "id": 3,
    "text": 6,
    "keyword": 4,
    "condition": 2,
    "cond_pfx": 5
  }
}
```

```
>>> mo, _id = ACT_LP.match('# * call `process_parts #'a3 :| #red')
>>> printf(mo.groups())
('', '* call `process_parts ', 'a3', ':|', '#red', None)
```

```
>>> mo, _id = ACT_LP.match('          something = more #'a1 :')
>>> printf(mo.groups())
('          ', 'something = more ', 'a1', ':', None, None)
```

```
>>> mo, _id = ACT_LP.match('          something = more #'a1 ::')
>>> if mo: printf(mo.groups())
```

```
>>> mo, _id = ACT_LP.match('          something = more #'a1 :-')
>>> if mo: printf(mo.groups())
```

```
>>> mo, _id = ACT_LP.match('          something = more #'a1 ;:')
>>> printf(mo.groups())
('          ', 'something = more ', 'a1', ':;', None, None)
```

```
>>> mo, _id = ACT_LP.match('rm -f "${top_dir}/.hgignore.new" #'a1 :')
>>> printf(mo.groups())
('', 'rm -f "${top_dir}/.hgignore.new" ', 'a1', ':', None, None)
```

```
>>> printf(ACT_RX.pattern)
^(\\s*)(?: (?://+|\\/+*+;+|@:u?[bl]?comm_?@|--|#+|[bl]?comm_?@[.][.]) )?(.*)\\s*(?://+|\\/+*+;+
↵+|/\\/+*+;+|@:u?[bl]?comm_?@|--|#+|[bl]?comm_?@[.][.]) ([[copduasmt][0-9]+) (:[:];|<>/})?|[:];
↵|<>/}.-)]\\s*(?: ([0-9A-Za-z]+|backwards) (?:\\s+|$))?(?: \\s(.*)|\\s*)$
```

```
>>> printf(ACT_LP.groups)
OrderedDict([('whitespace', 1), ('id', 3), ('text', 6), ('keyword', 4), ('condition', 2), (
↵'cond_pfx', 5)])
```

line_diversion.**check_line_parse** (*exprs)

Returns

prints

```
>>> check_line_parse('# s/^+//''p') #doctest: +ELLIPSIS
# -----
# ||:exp:|| # s/^+//p
# -----
# :DBG: ACT_LP      : ]--[ ]()[
# :DBG: ATTRIB_LP   : ]--[ ]()[
# :DBG: COND_LP     : ]--[ ]()[
# :DBG: PREFIX_LP   : ]--[ ]()[
# :DBG: PRINTE_SFORMAT_LP: ]--[ ]()[
# :DBG: PYJSMO_ATTRIB_LP : ]--[ ]()[
# :DBG: SUFFIX_LP   : ]--[ ]()[
```

```
>>> check_line_parse('# start #'a99') #doctest: +ELLIPSIS
# -----
# ||:exp:|| # start #a...99
# -----
# :DBG: ACT_LP      : ]--[ ]()[
# :DBG: ATTRIB_LP   : ]--[ ]()[
# :DBG: COND_LP     : ]--[ ]()[
# :DBG: PREFIX_LP   : ]--[ ]()[
# :DBG: PRINTE_SFORMAT_LP: ]--[ ]()[
# :DBG: PYJSMO_ATTRIB_LP : ]--[ ]()[
# :DBG: SUFFIX_LP   : ]a99[ ]('', 'start ', 'a99')[
```

```
>>> check_line_parse('.. start ..''a99') #doctest: +ELLIPSIS
# -----
# ||:exp:|| .. start ..a...99
# -----
# :DBG: ACT_LP      : ]--[ ]()[
# :DBG: ATTRIB_LP   : ]--[ ]()[
# :DBG: COND_LP     : ]--[ ]()[
# :DBG: PREFIX_LP   : ]--[ ]()[
# :DBG: PRINTE_SFORMAT_LP: ]--[ ]()[
# :DBG: PYJSMO_ATTRIB_LP : ]--[ ]()[
# :DBG: SUFFIX_LP   : ]a99[ ]('', 'start ', 'a99')[
```

```
>>> check_line_parse('start #'a99') #doctest: +ELLIPSIS
# -----
# ||:exp:|| start #a...99
# -----
# :DBG: ACT_LP      : ]--[ ]()[
# :DBG: ATTRIB_LP   : ]--[ ]()[
# :DBG: COND_LP     : ]--[ ]()[
# :DBG: PREFIX_LP   : ]--[ ]()[
# :DBG: PRINTE_SFORMAT_LP: ]--[ ]()[
# :DBG: PYJSMO_ATTRIB_LP : ]--[ ]()[
# :DBG: SUFFIX_LP   : ]--[ ]()[
```

```
class line_diversion.LineParser(*args, **kwargs)
```

```
>>> lp = LineParser()
>>> printf(lp)
{
  "rx": [
    null,
    "0"
  ],
  "groups": {
    "whitespace": null,
    "id": null,
    "text": null,
    "keyword": null,
    "condition": null
  }
}
```

```
>>> printf(lp._pyjsmo_x_rx)
None
```

```
>>> lp.rx = re.compile('some', re.I | re.M | re.U)
>>> printf(lp)
{
  "rx": [
    "some",
    "re.IGNORECASE|re.MULTILINE|re.UNICODE"
  ],
  "groups": {
    "whitespace": null,
    "id": null,
    "text": null,
    "keyword": null,
    "condition": null
  }
}
```

```
>>> printf(trans_rx_repr(lp._pyjsmo_x_rx)) #doctest: +ELLIPSIS
re.compile('some', re.IGNORECASE|re.MULTILINE|re.UNICODE)
```

```
>>> lp.rx = ("some where", "re.IGNORECASE|re.UNICODE")
>>> printf(lp)
{
  "rx": [
    "some where",
    "re.IGNORECASE|re.UNICODE"
  ],
  "groups": {
    "whitespace": null,
    "id": null,
    "text": null,
    "keyword": null,
    "condition": null
  }
}
```

```
>>> printf(trans_rx_repr(lp._pyjsmo_x_rx)) #doctest: +ELLIPSIS
re.compile('some where', re.IGNORECASE|re.UNICODE)
```

assemble (*part_map*, *diagram*)

Returns assembled line.

init (*rx*, *groups*)

Returns self for chaining.

match (*line*)

Match against line.

Returns match object or None.

rx

Programmatic property.

split (*line*, *mo*, *diagram*)

Returns

- part map
 - indent
 - text
 - kw
 - cond_pfx
 - condr
 - rest

- mapped_kw
- kw_sep
- cond
- kw_cont_sep
- mapped_kw_cont

split_ (*line, mo, diagram*)

Returns part map

class `line_diversion.DiagramTiming` (**args, **kwargs*)
 LineDiversion for Timing Diagram.

class `line_diversion.DiagramMessageSequence` (**args, **kwargs*)
 LineDiversion for Message Sequence Diagram.

class `line_diversion.DiagramInteraction` (**args, **kwargs*)
 LineDiversion for Interaction Diagram.

class `line_diversion.DiagramStateMachine` (**args, **kwargs*)
 LineDiversion for State Machine Diagram.

class `line_diversion.DiagramActivity` (**args, **kwargs*)
 LineDiversion for Activity Diagram.

class `line_diversion.DiagramUseCase` (**args, **kwargs*)
 LineDiversion for Use Case Diagram.

class `line_diversion.DiagramBehavior` (**args, **kwargs*)
 LineDiversion for Behavior Diagram.

class `line_diversion.DiagramDeployment` (**args, **kwargs*)
 LineDiversion for Deployment Diagram.

class `line_diversion.DiagramComponent` (**args, **kwargs*)
 LineDiversion for Component Diagram.

class `line_diversion.DiagramObject` (**args, **kwargs*)
 LineDiversion for Object Diagram.

class `line_diversion.DiagramClass` (**args, **kwargs*)
 LineDiversion for Class Diagram.

class `line_diversion.DiagramStructure` (**args, **kwargs*)
 LineDiversion for Structure Diagram.

class `line_diversion.Diagram` (**args, **kwargs*)

close_partition ()

Returns self for chaining.

open_partition (*text*)

Returns self for chaining.

process_parts (*part_map*)

Returns new/altered part_map.

```
>>> check_def = 'SomeClass ( with,multiple, inheritance )'
>>> mo = re.match(CLASS_DEF_RX, check_def)
>>> if mo: printf(sformat("{0}://{1}//", mo.groups(), check_def[mo.end(0):]))
('SomeClass', 'with,multiple, inheritance ')
```

```
>>> class Check():  
...     pass
```

```
>>> check_def = 'SomeClass ()'  
>>> mo = re.match(CLASS_DEF_RX, check_def)  
>>> if mo: printf(sformat("{0}://{1}//", mo.groups(), check_def[mo.end(0):]))  
('SomeClass', '')////
```

```
>>> check_def = 'SomeClass'  
>>> mo = re.match(CLASS_DEF_RX, check_def)  
>>> if mo: printf(sformat("{0}://{1}//", mo.groups(), check_def[mo.end(0):]))  
('SomeClass', None)////
```

```
class line_diversion.LineDiversion(*args,**kwargs)
```

```
close_partition()
```

Returns self for chaining.

```
finish()
```

Returns self for chaining.

```
open_partition(text)
```

Returns self for chaining.

RELEVANCE OF DOCUMENTATION

A map is not the territory it represents, but, if correct, it has a similar structure to the territory, which accounts for its usefulness. If the map could be ideally correct, it would include, in a reduced scale, the map of the map; the map of the map, of the map; and so on, endlessly, a fact first noticed by Royce.

—Alfred Korzybski, *Science and Sanity*, Chapter IV, p. 58.

The relevance of documentation, R_{doc} deteriorates, whenever the source code is changed, but the documentation is not updated.

The rate of deterioration for a given piece of documentation increases with

- the distance to the source code, $D_{source} \geq 0$
- the inavailability of a documentation editor, $I_{edit} \geq 0$
- the necessary effort for synchronization with the master document, $E_{sync} \geq 0$
- the inertia of the human components $H_{inertia} \geq 1$ involved.

The likeliness of relevance of a given piece of documentation, $L_{R_{doc}}$ is therefore:

$$L_{R_{doc}} = \left(\left(\frac{1 + (D_{source} + I_{edit} + E_{sync} + H_{inertia})}{2} \right) \cdot H_{inertia} \right)^{-1} \quad (8.1)$$

8.1 Introduction

The relevance of documentation is a measure of how similar the structure of the documentation is to the structure of the source code.

This is closely connected to the [map–territory relation](#) of [General semantics](#). A thorough understanding of this principle really helps manage the expectations of what documentation of source code is, is not and can be. The fact, that documentation can never be accurate in the sense of a bijective mapping, does not mean it can not be helpful – even essential – for a human brain to construct an appropriate model.

A major problem of software documentation arises from the fact, that the documented source code is often not available to the reader. Imagine, being given the following instructions for obtaining an item from a certain location:

- go straight
- go left
- look at the sign, that says “Menu”
- order the second item
- bring it back

However, you are only given access up to a fence with a locked gate, where you can see the straight part of the road and the left turn, but you cannot see the menu. At the locked gate, there is only a sign with yesterday’s menu. Now you are supposed to predict what the second item on today’s menu will be without being able to examine the actual menu (see [figure 8.1](#)). It is pretty obvious, that this cannot be reliably done, especially if the menu is changing daily to whatever the chef feels like.

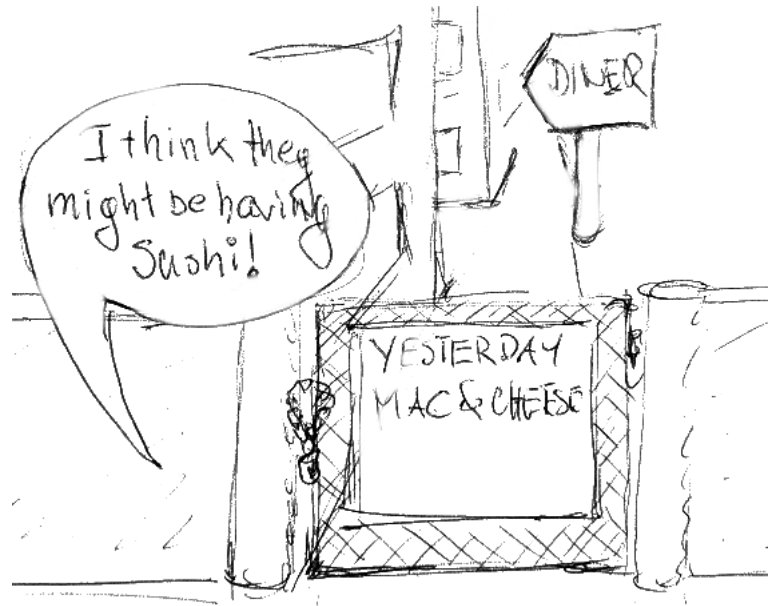


figure 8.1: I think they might be having sushi

This describes the situation developers often find themselves in when working with proprietary software. But even when source code access is available – since they are incidentally the developer, who works on it – they are still expected to produce documentation for people that cannot understand the code; i.e., these people – who may just as well be future versions of the developer himself – choose to stay at the fence and refuse to walk through the gate. So developers may write things like: “And then look around in the source code, to find out what the real state of affairs is at any given moment.” However, they cannot expect any particular reader to make the effort. See the [Sphinx](#) and [Doxygen](#) document generators in [section 14, Sphinx Documentation Generator](#) for excellent examples, how the entire source is included for cross-referencing¹.

So, how much interest can you muster for the utterly boring description of a treasure hunt for an unspecified treasure in a country you don’t know and don’t have the means to ever get to?

For me, that is one trivially self-evident answer to questions like “[What’s with the aversion to documentation in the industry?](#)”. The rapid deterioration of relevance for documentation during the development phase being another.

However, having no documentation at all is not a viable option and there are certainly ways to reduce the effort and increase the relevance of documentation.

8.2 Source and documentation management

Derived from the above equation, the best likeliness of relevance of a given piece of documentation is achieved when all deterioration factors, D_{source} , I_{edit} , E_{sync} , $H_{inertia}$ are at a minimum, which makes $L_{R_{doc}} = 1$.

To discuss the likeliness of relevance of a given piece of documentation, it is necessary to define a model, which explains the parameters in their respective context (see [figure 8.2](#)).

Note: The required editor for a change is just one of a possible multitude of appropriate editors. It is not necessarily the preferred editor of a specific programmer (see [figure 8.3](#)).

¹ [Sphinx](#) even includes the source code for the documentation itself. However, the [map-territory relation](#) still holds, since there is no description of the process that transforms the documentation source code into the final output.

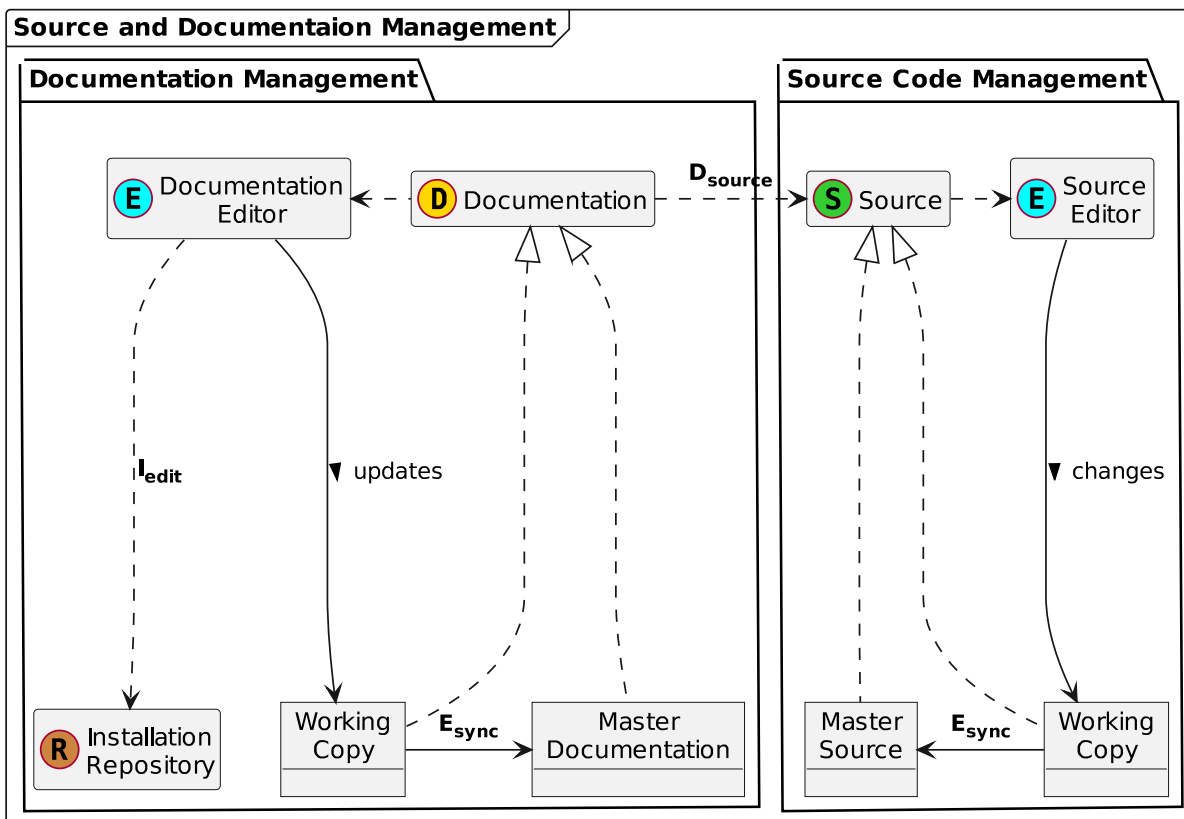


figure 8.2: Source and documentation management

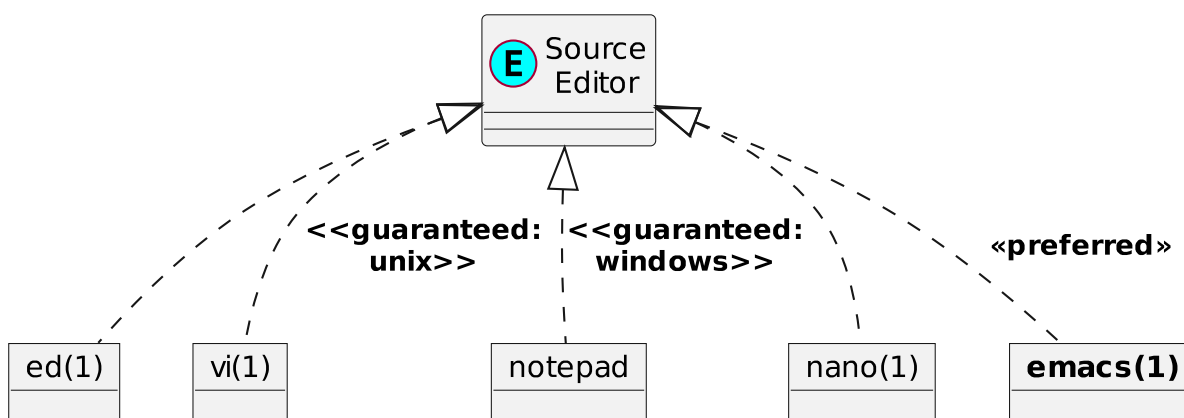


figure 8.3: Guaranteed and preferred editors

8.3 Minimum Distance to Source Code

A distance to source code, D_{source} of 0 corresponds to the principle “**the source is the documentation**”, i.e., there is no additional documentation for the source code. Substituting D_{source} by 0 in equation (8.1) results in:

$$\left(\left(\frac{1 + (I_{edit} + E_{sync} + H_{inertia})}{2} \right)^{H_{inertia}} \right)^{-1} \quad (8.2)$$

In this case, the source editor and the documentation editor are necessarily the same and therefore I_{edit} can be eliminated from equation (8.2):

$$\left(\left(\frac{1 + (E_{sync} + H_{inertia})}{2} \right)^{H_{inertia}} \right)^{-1} \quad (8.3)$$

$$L_{Rdoc} = (1 + (D_{source} + E_{sync})^{H_{inertia}})^{-1}$$

When the source code changes are made to the master source file, E_{sync} is also 0 and equation (8.3) is reduced to:

$$\left(\left(\frac{1 + H_{inertia}}{2} \right)^{H_{inertia}} \right)^{-1} \quad (8.4)$$

Without automatic documentation generators that have to be triggered manually, human inertia $H_{inertia}$ is 1 and equation (8.4) is reduced to 1:

$$\left(\left(\frac{1 + 1}{2} \right)^1 \right)^{-1} = 1 \quad (8.5)$$

So it seems, that the only time, documentation is guaranteed to be entirely relevant, is if there is none. In all other cases, documentation and source strictly abide by the properties of the [map–territory relation](#)².

However, there are tools like `grep(1)`, `ctags(1)`, `etags(1)`, `cscope(1)`, even [Sphinx](#) and [doxygen](#), also (reluctantly) IDEs like Eclipse, which extract meta-information from undocumented source code to provide hyper linking facilities, which is in itself an extremely useful part of any documentation.

Other Generators produce Nassi-Shneiderman diagrams (flowcharts, activity diagrams), dependency graphs, etc.

Therefore the principle “**the source is the documentation**” is actually not so blunt or ridiculous as it may sound at first.

However, since automatically generated documentation requires a compilation step that may require manual triggering, $H_{inertia}$ may quickly become a significant factor.

8.4 Inertia

In the eternal sunshine of the spotless mind, each relevant source modification triggers a documentation update (see [figure 8.4](#)).

But this is only true, when no human components are involved in the process of deciding, whether a source change is relevant for a documentation update. And this can only be the case when no manually created documentation parts have to be maintained. An example for this is the initial documentation produced by generators like [Sphinx](#) and [Doxygen](#).

However, there is usually a more or less complex update decision process involved, which potentially results in increases of I_{edit} and human inertia $H_{inertia}$ (see [figure 8.5](#)).

The decision process in state *Update Decision* looks somewhat like the activity diagram in [figure 8.6](#). Feel free to add an infinite number of excuses for not updating the documentation or postponing the update indefinitely.

² Considering that the source code of a program is itself a map of an executable, requiring a processor (interpreter) and possibly a compiler to produce another representation, there really is no simple escape from the [map–territory relation](#).

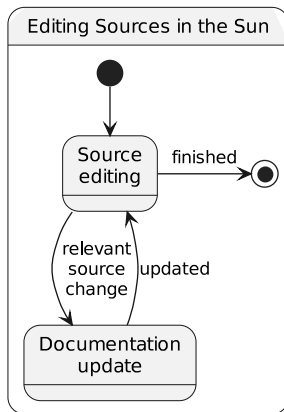


figure 8.4: Sunny day documentation update

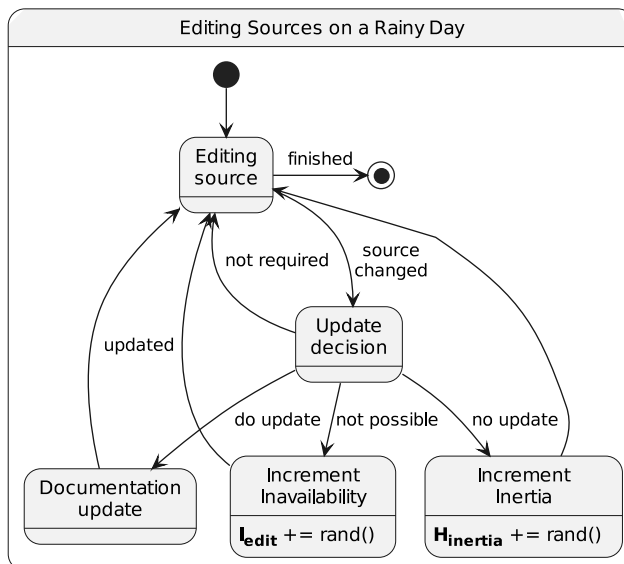


figure 8.5: Rainy day documentation update

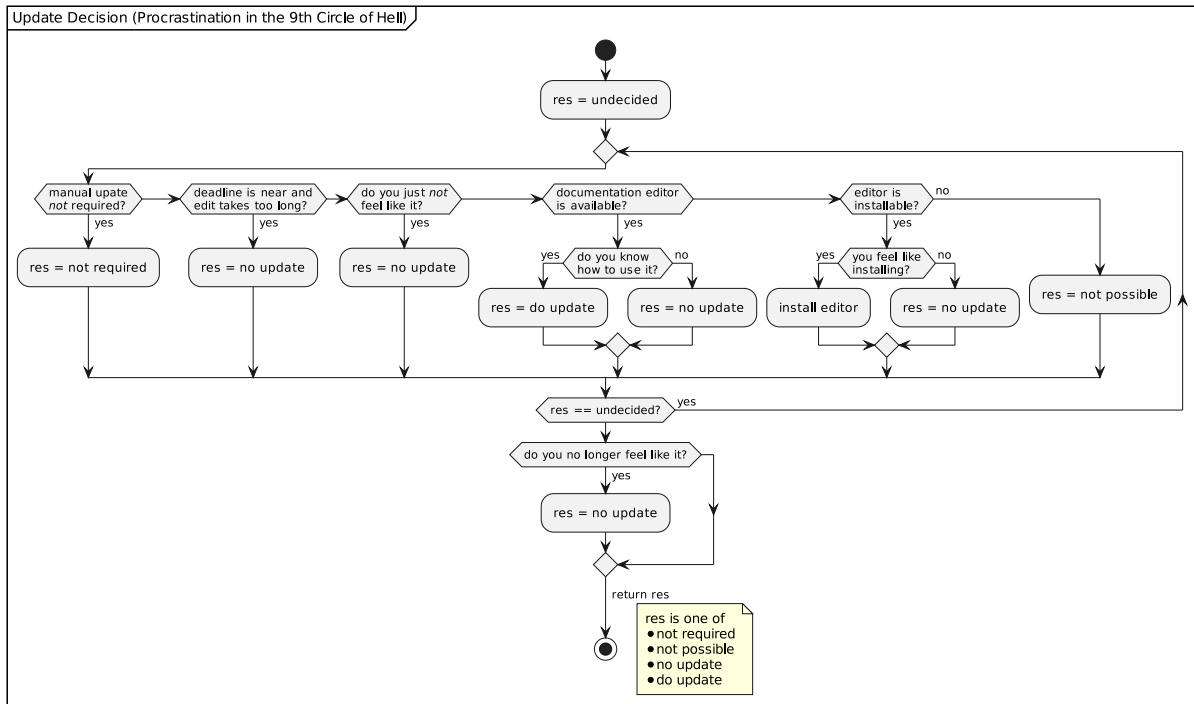


figure 8.6: Documentation update decision

8.5 Single Editor for Source and Documentation

When the documentation editor is the same as the source editor, the decisions about using or installing a separate documentation editor are removed. Therefore the result “**not possible**” is no longer returned (see figure 8.7).

Consequently, the state machine for editing sources is also reduced. The state “**Increment Inavailability**” is removed and therefore I_{edit} is no longer incremented (see figure 8.8).

Therefore I_{edit} is always 0 and equation (8.1) is reduced to:

$$L_{R_{doc}} = \left(\left(\frac{1 + (D_{source} + E_{sync} + H_{inertia})}{2} \right) \cdot H_{inertia} \right)^{-1} \quad (8.6)$$

Note: Please, note, that in addition to a documentation editor, there may be separate documentation generators required to produce some form of final documentation (e.g., SVG, PNG, JPEG, HTML, PDF). However, the lack of these does not contribute to the deterioration of relevance, as long as the documentation source is updated.

If it is found, that the update of a piece of documentation without a visual representation created by the documentation generator is not possible, this decision may have to be revisited.

8.6 Synchronization

When there is a single file on just one computer, there is obviously no synchronization necessary and therefore $E_{sync} = 0$.

Sometimes it is better to copy a project directory and work on the copy so that others are not disturbed by the intermediate states of development. That is called a *branch* or *working on a branch*. The original project file is the *master*.

As long as nothing leaves the branch directory, the documentation is not affected by the effort necessary for synchronizing the branch with the master. It is only, when e.g. a binary version from the branch is delivered to a

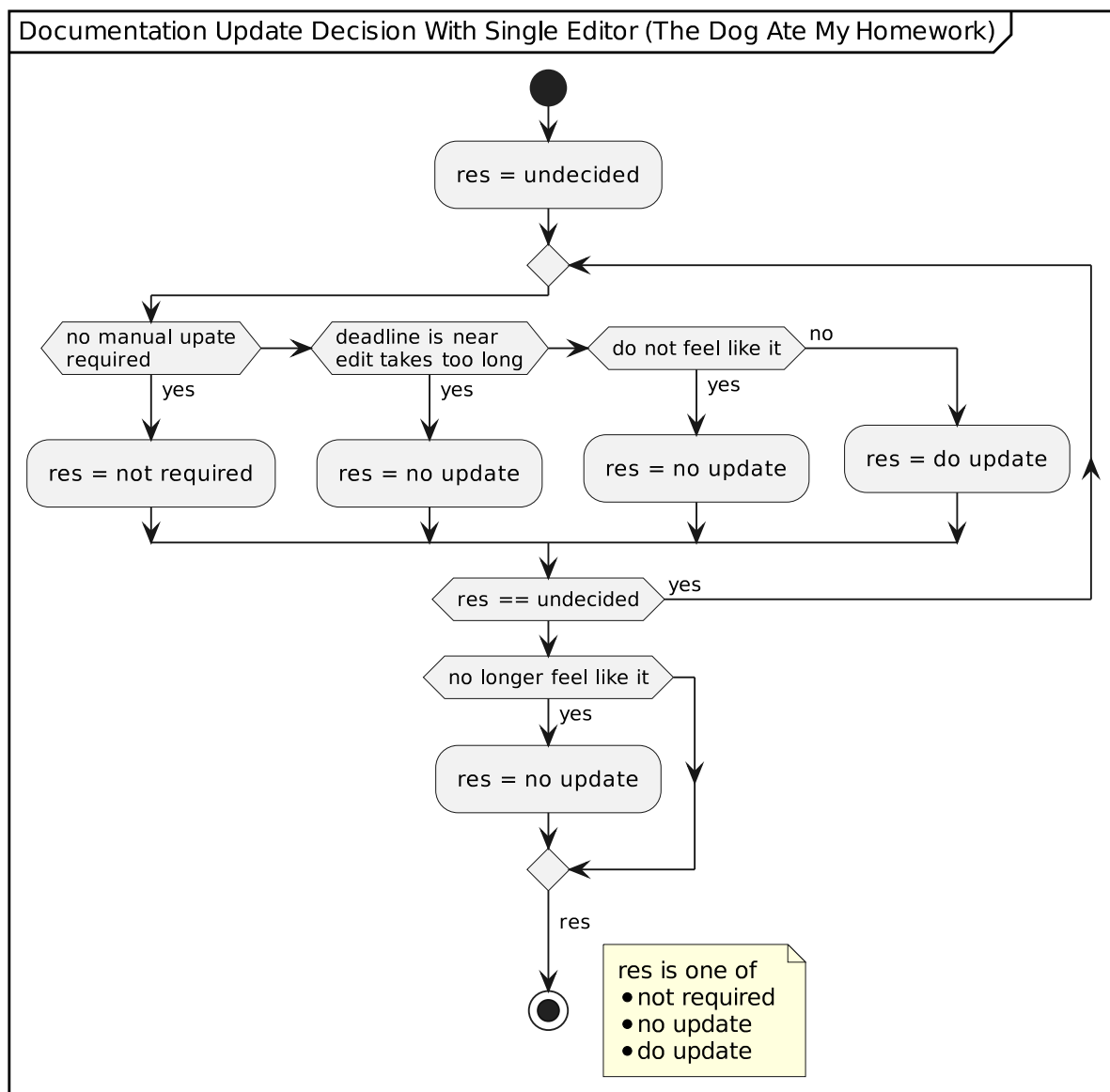


figure 8.7: Documentation Update Decision With Single Editor

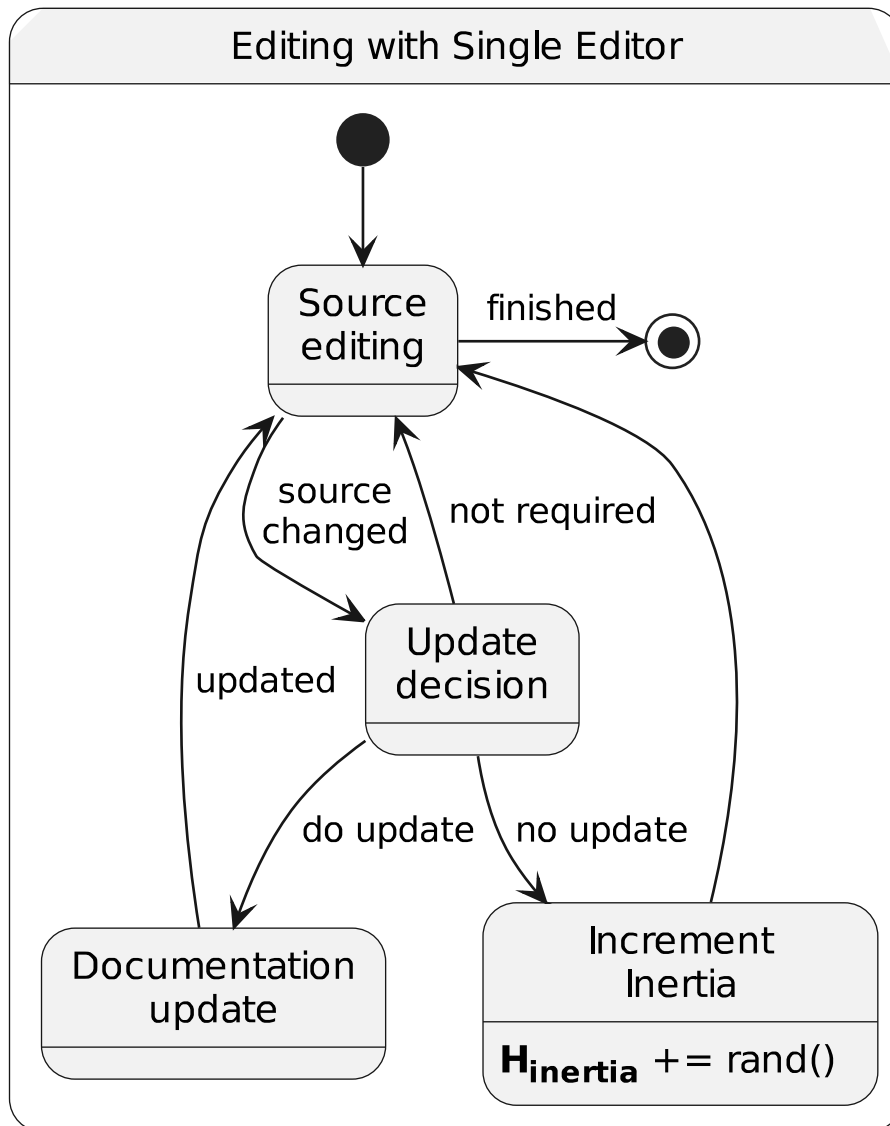


figure 8.8: Editing with single editor

production system, that the master documentation becomes less relevant until the branch with all documentation changes is synchronized (*merged*) with the master. This is the situation when E_{sync} becomes > 0 .

Human forgetfulness and general sloppiness are major factor for discrepancies, where branches are left to rot on flash drives and, newer versions are overwritten by older versions, etc.

This is why the process is nowadays highly automated by distributed version control systems (DVC).

DVCs are quite good, but sometimes you don't want to check in changes, that need some final touch. In that case, you are back to copy or remote copy (`scp(1)`, `rsync(1)`). So all other stages are still useful:

- backup file
- `diff(1)`, `patch(1)`, `diff3(1)`
- `scp(1)`, `rsync(1)`
- VCS
- DVCS

8.6.1 qs-gen-sync.pl (.sync.rc)

There are two aliases for *backup* and *restore*:

```
bsy send stuff to remote host ./sync.sh --backup
```

```
rsy get stuff from remote host ./sync.sh --restore
```

Generate initial `.sync.rc`:

```
isy -n
```

8.6.2 diff3

!todo! finish description of `diff3(1)`

```
diff3 doc/index.rst README.txt doc/overview.rst
```

8.7 Battling Human Inertia

The only way to defeat human inertia, $H_{inertia}$, is automation.

See also [figure 8.9](#)

!todo! finish description of scripting `sh(1)`, `make(1)`

- `make/scripts` are memory banks for knowledge

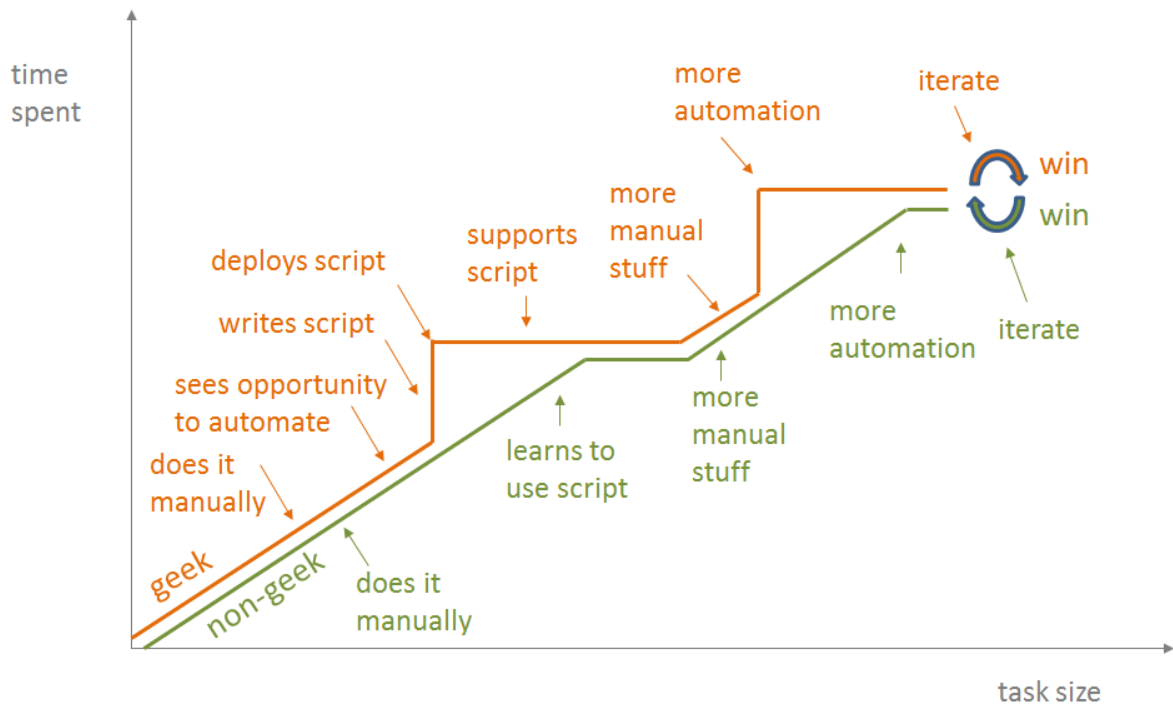
8.7.1 Integrated Development Environment

An IDE strives to be an *integrated* solution for cross-referencing, build automation, generated documentation.

In that sense, `emacs(1)` can also be classified as an IDE.

Eclipse is ridiculously hard to extend out of the box with plug-ins written in Java. E.g., in a forum question from 2016 about [how to extend the default Java editor](#), somebody just wants to add a few keywords to the Java editor and gets links to a [PDF slideshow about JDT](#) and to `Xtext`. Neither are anywhere near trivial.

EASE does provide integration of script engines into Eclipse giving access to the workbench. This engine may be able to handle couple of keywords, if it can be added to Eclipse extension points (hooks), which needs to be



Source: Another way to think about geeks and repetitive tasks - Jon Udell

figure 8.9: Geeks and repetitive tasks

researched **!todo!**. So in 2017 eclipse announces another convoluted overstructured object oriented integration, which still does not reach easily into an editing buffer (see also [Internal and External Editors](#)).

See also <https://stackoverflow.com/questions/26912785/eclipse-jdt-syntax-highlighting-of-constants> from 2014, which is still unanswered. The answers in <https://stackoverflow.com/questions/13802131/in-an-eclipse-plugin-how-can-i-programmatically-highlight-lines-of-codes-in-the>, give a variety of strategies to follow, but none of the answers gives a step-by-step example.

Here is another failed attempt <https://www.eclipse.org/forums/index.php/t/489221/> for adding some highlighting to the Eclipse editor.

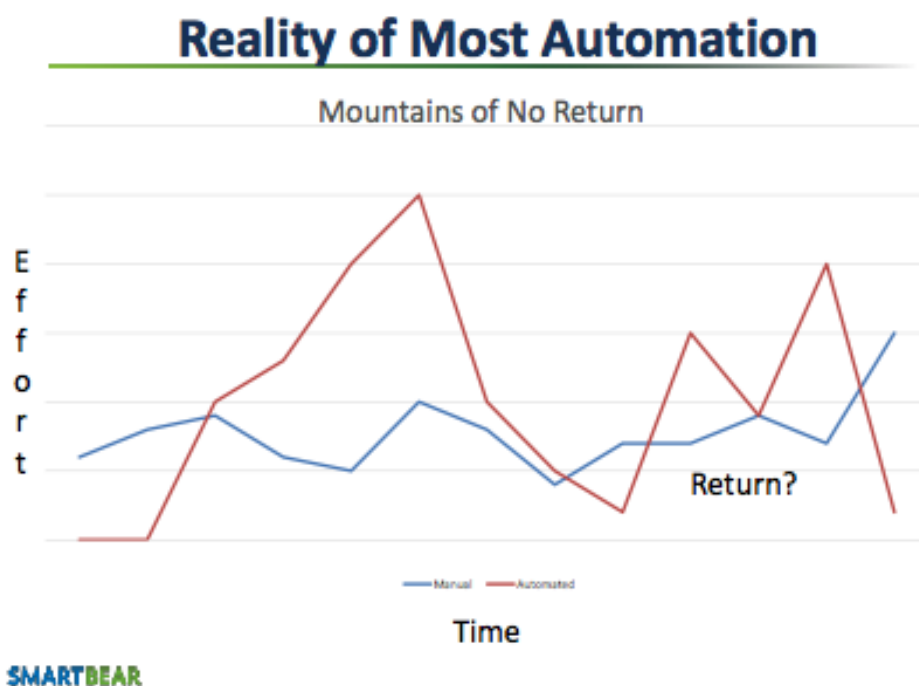
All in all Eclipse has still not reached the efficient state of the art provided by emacs(1) for decades. emacs(1) can be extended ridiculously simple with a multitude of tools glued together by Elisp.

As discussed in the article [Using Automation to Assist – not replace – Manual Testing](#), automating entire workflows is very much ineffective (see figure 8.10).

In that context, a tightly integrated IDE – like Eclipse or Visual Studio – is the ineffective attempt to provide an entirely automated workflow in a one-size-fits-all manner.

In contrast, the loosely integrated approach with emacs(1) and tools provides a framework for task automation which can adapt to a variety of workflows.

A monolithic IDE usually has a plethora of cryptic configuration settings to provide some sort of rudimentary flexibility (customization), whereas the tool based IDE allows full modification of each automation step down to the code level.



Source: Using Automation to Assist – not replace – Manual Testing

figure 8.10: Using Automation to assist

VERSION CONTROL SYSTEM

A version control system directly affects the synchronization effort E_{sync} .

9.1 Cherry-Picking

9.1.1 Mercurial

```
cd repo
hg export --git -r REV >000-patch.dif
hg export --git -r 685 >000-patch.dif
```

Adding additional changes to the patch

1. clone repository

```
cd ..
hg clone $( pwd )/repo repo-cherry
```

2. import the patch

```
cd repo-cherry
hg import ../repo/000-patch.dif
```

3. modify sources

4. create new patch

```
sed '1,/^$/p;d' ../repo/000-patch.dif >../repo/001-patch.dif
hg diff --git -r -2 >>../repo/001-patch.dif
```

5. re-create clone, apply final patch

```
cd ..
rm -rf repo-cherry
hg clone $( pwd )/repo repo-cherry

cd repo-cherry
hg import ../repo/001-patch.dif
```

After testing, the patch 001-patch.dif can be applied to the clean master repository.

Note: If an import fails, added files are created, but not added to the repository. This must be done manually.

9.1.2 Git

:todo: git cherry-picking

VCS - MERCURIAL

10.1 Repository Manipulation

Check out *graft*, *transpose*.

10.1.1 convert

Make sure, that there are no modifications in source repository:

```
hg -R /home/ws/project/ws_rfid/demo status -q
```

Get filemap of active files:

```
hg -R /home/ws/project/ws_rfid/demo status -c | sed 's,^C,include,' >filemap.txt
```

Convert source repo to new repo without deleted files:

```
hg convert --filemap filemap.txt /home/ws/project/ws_rfid/demo demo
```

Update new repo:

```
( cd demo && hg update -C )
```

```
dest_repo="adhoc-c"
rm -rf "${dest_repo}"
cat <<'EOF' >filemap
include "adhoc"
rename "adhoc" "."
EOF
hg convert --filemap filemap /home/ws/project/ws-util "${dest_repo}"
(
cd "${dest_repo}" || exit 1
hg update -C
)
```

10.1.2 Remove unwanted branches, clone -rev

Remove unwanted branches:

```
hg clone --rev xxx URL
```

Also copy `.hg/hgrc`!

11.1 GUI

The standard GUI for git(1) is:

```
gitk  
gitk --all
```

The commit tool is:

```
git gui
```

Other programs:

cola The highly caffeinated Git GUI

qgit QT interface to git trees, with stgit support.

gitg Git repository viewer

11.2 github fork

After *cloning a github fork* and *adding an upstream repository*, everyday repeating operation is *syncing the github fork with upstream*.

11.2.1 Syncing the github fork with upstream

1. Clean up:

```
git stash  
git checkout master # -f
```

2. Update upstream:

```
git fetch upstream
```

3. Merge upstream:

```
git merge upstream/master
```

4. Get current work:

```
git stash pop / apply  
git stash clear # if anything left on stash
```

11.2.2 Cloning a github fork

```
git clone git@github.com:wolfmanx/REPOSITORY.git
```

11.2.3 Adding an upstream repository

```
git remote add upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git
```

Example:

```
sheckley:~/3rd-party/w2ui/w2ui$ git remote -v
origin      git@github.com:wolfmanx/w2ui.git (fetch)
origin      git@github.com:wolfmanx/w2ui.git (push)
upstream    https://github.com/vitmalina/w2ui.git (fetch)
upstream    https://github.com/vitmalina/w2ui.git (push)
```

11.3 Quickstart

See status:

```
git status
```

Get updated info from remote repositories:

```
git fetch --all
```

Get and merge remote into local:

```
git pull
```

Show branches:

```
git branch
git branch -a
```

Update remote refs along with associated objects:

```
git push
```

Show remotes:

```
git remote -v
```

Go to specific commit (revision):

```
git checkout revision
```

Fix # HEAD detached at 1.7.1:

```
git checkout master -f
```

If you want to revert changes made to your working copy, do this:

```
git checkout . -f
```

If you want to revert changes made to the index (i.e., that you have added), do this:

```
git reset
```

If you want to revert a change that you have committed, do this:

```
git revert ...
```

Abort a merge:

```
git reset --hard HEAD
```

Remove changes:

```
git branch  
git reset --hard HEAD
```

Create/apply patches:

```
git format-patch deadbeef..b00b5 # => 001-title.patch, 002-title.patch, ...  
git format-patch master.. # everything on a branch  
  
git format-patch master...branch # from merge point  
  
git am *.patch # separate commits  
# if there are problems  
git am --continue  
git am --skip  
git am --abort  
  
git apply *.patch # no commit, single patch
```

11.4 Tricks

11.4.1 Find the first commit of a branch

```
git log master..branch --oneline | tail -1
```

E.g.:

```
git log master..i18n-default-locale --oneline | tail -1  
# 40f5c99 option i18n.notrans for language alias of C locale  
  
git diff 40f5c99~1 i18n-default-locale >tg2-i18n-notrans.patch
```

11.4.2 Delete commits

See [git rebase - Delete commits from a branch in Git - Stack Overflow](#)

Assuming you are sitting on that commit, then this command will wack it...

```
git reset --hard HEAD~1
```

The HEAD~1 means the commit before head.

Or, you could look at the output of git log, find the commit id of the commit you want to back up to, and then do this:

```
git reset --hard <sha1-commit-id>
```

If you already pushed it, you will need to do a force push to get rid of it...

```
git push origin HEAD --force
```


However, if others may have pulled it, then you would be better off starting a new branch. Because when they pull, it will just merge it into their work, and you will get it pushed back up again.

If you already pushed, it may be better to use `git revert`, to create a “mirror image” commit that will undo the changes. However, both commits will be in the log.

FYI—`git reset --hard HEAD` is great if you want to get rid of WORK IN PROGRESS. It will reset you back to the most recent commit, and erase all the changes in your working tree and index.

Lastly, if you need to find a commit that you “deleted”, it is typically present in `git reflog` unless you have garbage collected your repository.

11.4.3 How to modify existing, unpushed commits

Best answer on [Stack Overflow](#) is: [git - How to modify existing, unpushed commits?](#):

Question:

I wrote the wrong thing in a commit message. Alternatively, I’ve forgotten to include some files.

How can I change the commit message/files? The commit has not been pushed yet.

Amending the most recent commit message

```
git commit --amend
```

will open your editor, allowing you to change the commit message of the most recent commit. Additionally, you can set the commit message directly in the command line with:

```
git commit --amend -m "New commit message"
```

However, this can make multi-line commit messages or small corrections more cumbersome to enter.

Make sure you don’t have any working copy changes staged before doing this or they will get committed too. (Unstaged changes will not get committed.)

Changing the message of a commit that you’ve already pushed to your remote branch

If you’ve already pushed your commit up to your remote branch, then you’ll need to force push the commit with:

```
git push <remote> <branch> --force
# Or
git push <remote> <branch> -f
```

Warning: force-pushing will overwrite the remote branch with the state of your local one. If there are commits on the remote branch that you don’t have in your local branch, you will lose those commits.

Warning: be cautious about amending commits that you have already shared with other people. Amending commits essentially rewrites them to have different SHA IDs, which poses a problem if other people have copies of the old commit that you’ve rewritten. Anyone who has a copy of the old commit will need to synchronize their work with your newly re-written commit, which can sometimes be difficult, so make sure you coordinate with others when attempting to rewrite shared commit history, or just avoid rewriting shared commits altogether.

Use interactive rebase

Another option is to use interactive rebase. This allows you to edit any message you want to update even if it’s not the latest message.

In order to do a git squash, follow these steps:

```
# X is the number of commits to the last commit you want to be able to edit
git rebase -i HEAD~X
```

Once you squash your commits - choose the e/r for editing the message

Important note about Interactive rebase

When you use the `git rebase -i HEAD~X` there can be more than X commits. Git will “collect” all the commits in the last X commits and if there was a merge somewhere in between that range you will see all the commits as well so the outcome will be X+.

Good tip:

If you have to do it for more than a single branch and you might face conflicts when amending the content set up the `git rerere` and let git resolve those conflicts automatically for you.

11.5 Create branches

Branching allows you to build new features or test out ideas without putting your main project at risk. In git, branch is a sort of bookmark that references the last commit made in the branch. This makes branches very small and easy to work with.

How do I use branches?

Branches are pretty easy to work with and will save you a lot of headaches, especially when working with multiple people. To create a branch and begin working in it, run these commands:

```
git branch mybranch
# Creates a new branch called "mybranch"

git checkout mybranch
# Makes "mybranch" the active branch
```

Alternatively, you can use the shortcut:

```
git checkout -b mybranch
# Creates a new branch called "mybranch" and makes it the active branch
```

Create a branch tracking a remote branch, if it was created remotely:

```
git checkout -b serverfix origin/serverfix
```

Or push it, when ready:

```
git push <remote-name> <branch-name>
```

Where `<remote-name>` is typically origin, the name which git gives to the remote you cloned from. Note however that formally, the format is:

```
git push <remote-name> <local-branch-name>:<remote-branch-name>
```

To switch between branches, use `git checkout`:

```
git checkout master
# Makes "master" the active branch

git checkout mybranch
# Makes "mybranch" the active branch
```

Once you're finished working on your branch and are ready to combine it back into the master branch, use `merge`:

```
git checkout master
# Makes "master" the active branch

git merge mybranch
# Merges the commits from "mybranch" into "master"

git branch -d mybranch
# Deletes the "mybranch" branch
```

Tip: When you switch between branches, the files that you work on (the “working copy”) are updated to reflect the changes in the new branch. If you have changes you have not committed, git will ensure you do not lose them. Git is also very careful during merges and pulls to ensure you don’t lose any changes. When in doubt, commit early and commit often.

11.6 Patches

See [Create a git patch from the changes in the current working directory - Stack Overflow](#):

```
git format-patch master..my-branch
```

See [How to apply a patch generated with git format-patch? - Stack Overflow](#)

Note: You can first preview what your patch will do:

First the stats:

```
git apply --stat a_file.patch
```

Then a dry run to detect errors:

```
git apply --check a_file.patch
```

Finally, you can use `git am` to apply your patch as a commit: it allows you to sign off an applied patch.

This can be useful for later reference.

```
git am --signoff < a_file.patch
```

11.7 Github Forking

11.7.1 Step 3: Configure remotes

When a repository is cloned, it has a default remote called origin that points to your fork on GitHub, not the original repository it was forked from. To keep track of the original repository, you need to add another remote named upstream:

```
cd Spoon-Knife
# Changes the active directory in the prompt to the newly cloned "Spoon-Knife" directory

git remote add upstream https://github.com/octocat/Spoon-Knife.git
# Assigns the original repository to a remote called "upstream"

git fetch upstream
# Pulls in changes not present in your local repository, without modifying your files
```

More Things You Can Do

You've successfully forked a repository, but get a load of these other cool things you can do: Push commits

Once you've made some commits to a forked repository and want to push it to your forked project, you do it the same way you would with a regular repository:

```
git push origin master
# Pushes commits to your remote repository stored on GitHub
```

Pull in upstream changes

If the original repository you forked your project from gets updated, you can add those updates to your fork by running the following code:

```
git fetch upstream
# Fetches any new changes from the original repository

git merge upstream/master
# Merges any changes fetched into your working files
```

What is the difference between fetch and pull?

There are two ways to get commits from a remote repository or branch: `git fetch` and `git pull`. While they might seem similar at first, there are distinct differences you should consider.

Pull:

```
git pull upstream master
# Pulls commits from 'upstream' and stores them in the local repository
```

When you use `git pull`, `git` tries to automatically do your work for you. It is context sensitive, so `git` will merge any pulled commits into the branch you are currently working in. One thing to keep in mind is that `git pull` automatically merges the commits without letting you review them first. If you don't closely manage your branches you may run into frequent conflicts.

Fetch & Merge:

```
git fetch upstream
# Fetches any new commits from the original repository

git merge upstream/master
# Merges any fetched commits into your working files
```

When you `git fetch`, `git` retrieves any commits from the target remote that you do not have and stores them in your local repository. However, it does not merge them with your current branch. This is particularly useful if you need to keep your repository up to date but are working on something that might break if you update your files. To integrate the commits into your local branch, you use `git merge`. This combines the specified branches and prompts you if there are any conflicts.

11.8 Resolving merge conflicts with git and kdiff3

From <http://jeroen.haeghebaert.com/post/2008/08/26/Resolving-merge-conflicts-with-git-and-kdiff3>

To fix the conflicts using `kdiff3`, first you need to tweak the `kdiff3` configuration a bit: start `kdiff3`, and go to Settings, Configure KDiff3, Integration. Under 'Command line options to ignore', add '-' separated by a ';' if necessary. To resolve conflicts, you can now use:

```
git mergetool -t kdiff3
```

To permanently configure `kdiff3` as the merge tool (so you don't need to specify the '-t kdiff3' anymore):

```
git config merge.tool kdiff3
```

UNIFIED MODELING LANGUAGE

12.1 UML Introduction

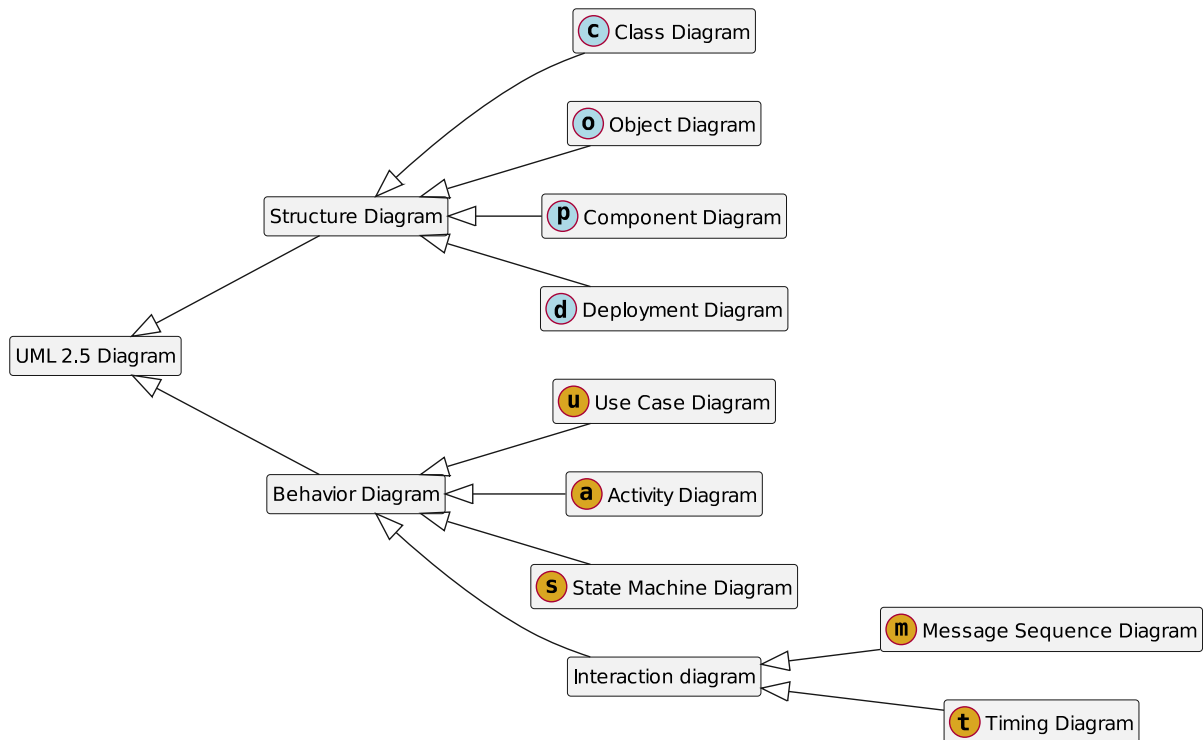
UML (Unified Modeling Language) is an ISO standardized method for describing programs. Its resulting diagrams are widely taught and should therefore be used as much as possible. The standard is mostly about the inner mechanism of the modeling language and not so much about the diagrams, however, the diagrams are the most useful part, as they are quickly understood even by non-programmers.

12.1.1 Getting Started with UML Diagrams

Check out these web-sites

- [Unified Modeling Language \(UML\) description](#)
The most comprehensive reference for UML 2.5 diagrams
- [Sparx Systems - UML 2 Tutorial](#)
A concise overview of UML 2 diagrams

This is the inheritance hierarchy of UML diagrams supported by [PlantUML](#) (clickable in HTML):



- [UML 2.5 Diagram](#)

- Structure Diagram
 - * Class Diagram
 - * Object Diagram
 - * Component Diagram
 - * Deployment Diagram
- Behavior Diagram
 - * Use Case Diagram
 - * Activity Diagram
 - * State Machine Diagram
 - * Interaction Diagram
 - Message Sequence Diagram
 - Timing Diagram

12.1.2 Quick Reference Guides

- [Local copy of from UML 2.5 Referenz - Karte, Diagramm, Notation](#)
- [Local copy of PDQM UML Quick Reference Guide](#)
- [Allen Holub's UML Quick Reference](#)
- [Local copy of UML Quick Reference Guide from No Magic.](#)
- [Local copy of UML Cheatsheet from Lou Franco: code, apps, and writings](#)

12.1.3 Reference Guides and Tutorials

- [Practical UML: A Hands-On Introduction for Developers](#)
- [Microsoft - Create models for your app](#)
- [tutorialspoint - UML Tutorial \(old\)](#)

12.1.4 UML Tools

The site [UML tools : Curated selection of free, online, OSS, for MAC, . . . tools](#) is an extensive collection of UML tool information.

12.1.5 UML GUI Tools

Many UML tools specialize in providing a GUI environment to design diagrams with extensive support for a detailed standards-compliant specification of elements. The output is standardized but unreadable XMI (XML Metadata Interchange). This disqualifies most of the UML GUI tools for proper adhoc ASCII level documentation.

Creating unreadable diagram definitions, which require HTML/PDF/image generation before they become useful is highly undesirable if your focus is on programming. The incentive for a programmer to keep the documentation up to date is negative¹.

GUI Tools are still useful to explore the graphical notation of the various diagrams. See [ref:gui tools with uml standard support](#).

¹ As long as programmers maintain documentation, it will always be more or less out of date. When documentation is decoupled from a program it is nothing but guaranteed to become incorrect.

12.1.6 Tools for Translating Declarative Specs to Diagrams

If your focus is on modeling, the *UML GUI Tools* are a better choice, but if your main task is programming, this is the way to go. The introduction to *UMLGraph* provides an excellent argument for declarative specification of UML diagrams.

The *Stack Overflow* question “Generating UML diagrams from textual representation” references a list of tools (*Text to UML tools*), but highlights *PlantUML*.

The *PlantUML* tool’s definition language is quite readable without the rendered diagrams. It also integrates with many tools (namely **Emacs** and *Sphinx*)- (See section *Sequence Diagram*). The GitHubGist *generate PlantUML definition from python sources* provides `genclass.py`, locally renamed to `gen_plantuml.py`. This command helps with the initial UML class diagrams and further updates. See section *Auto generators* for an example generator of a class diagram for global variables/functions.

yUML has *sphinx* integration, but is very limited in regarding readability and choice of diagrams (see section *yUML*).

Umple: Merging Modeling with Programming, i.e. it works as a programming language pre-processor (programming language extension, *roundtrip*).

12.2 UML Diagrams

The most important UML Diagrams are

- Use Case Diagram
- Class Diagram
- Activity Diagram
- State Machine Diagram
- *Sequence Diagram*

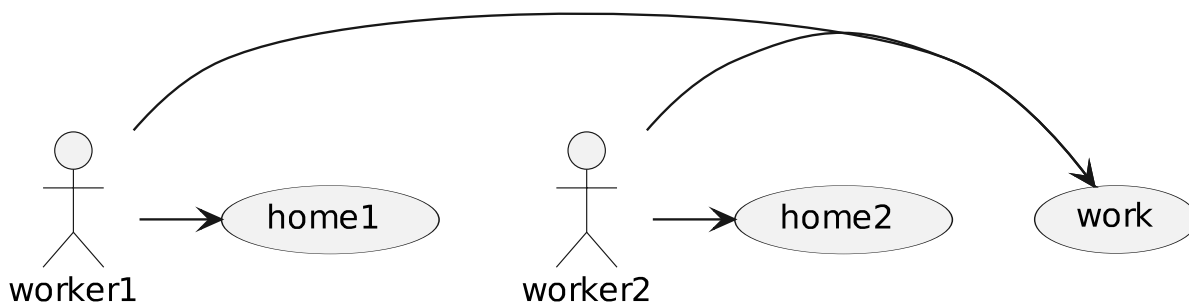
12.2.1 Use Case Diagram

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system. (<https://www.uml-diagrams.org/use-case-diagrams.html>)

Here is the use case diagram of a workers day:

```
.. uml::
    @startuml
    worker1 -> (work)
    worker2 -> (work)
    worker1 -> (home1)
    worker2 -> (home2)
    @enduml
```

This is how it is rendered in *Sphinx*:



For details see [PlantUML Language Reference Guide](#).

12.2.2 Class Diagram

Class diagram is UML structure diagram which shows structure of the designed system at the level of classes and interfaces, shows their features, constraints and relationships - associations, generalizations, dependencies, etc. (<https://www.uml-diagrams.org/class-diagrams-overview.html>)

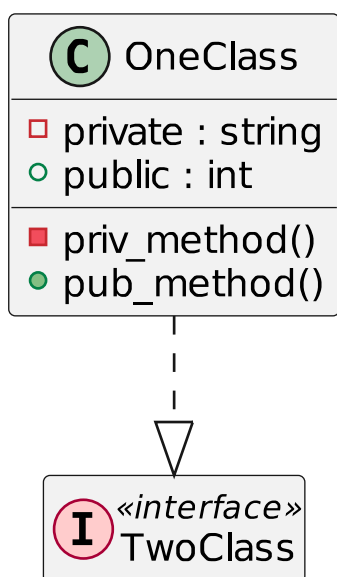
PlantUML example ([PlantUML Language Reference Guide](#)):

```

.. uml::

@startuml
class OneClass {
  -private : string
  +public : int
  -priv_method()
  +pub_method()
}
'stereotype
class TwoClass << (I, #ffcccc) interface >>
OneClass ..|> TwoClass
hide TwoClass members
@enduml
  
```

This is how it is rendered in Sphinx:



Associations between classes

UML association is relationship between classifiers to show that instances of classifiers could be either linked to each other or combined into some aggregation. See [the definitive Guide to UML associations](#).

Association is a relationship where all objects have their own lifecycle and there is no owner.

Let's take an example of Teacher and Student. Multiple students can associate with single teacher and single student can associate with multiple teachers, but there is no ownership between the objects and both have their own lifecycle. Both can be created and deleted independently.

Aggregation is a specialised form of Association where all objects have their own lifecycle, but there is ownership and child objects can not belong to another parent object.

Let's take an example of Department and teacher. A single teacher can not belong to multiple departments, but if we delete the department, the teacher object will not be destroyed. We can think about it as a "has-a" relationship.

Composition is again specialised form of Aggregation and we can call this as a "death" relationship. It is a strong type of Aggregation. Child object does not have its lifecycle and if parent object is deleted, all child objects will also be deleted.

Let's take again an example of relationship between House and Rooms. House can contain multiple rooms - there is no independent life of room and any room can not belong to two different houses. If we delete the house - room will automatically be deleted.

Let's take another example relationship between Questions and Options. Single questions can have multiple options and option can not belong to multiple questions. If we delete the questions, options will automatically be deleted.

Source: [StackExchange](#)

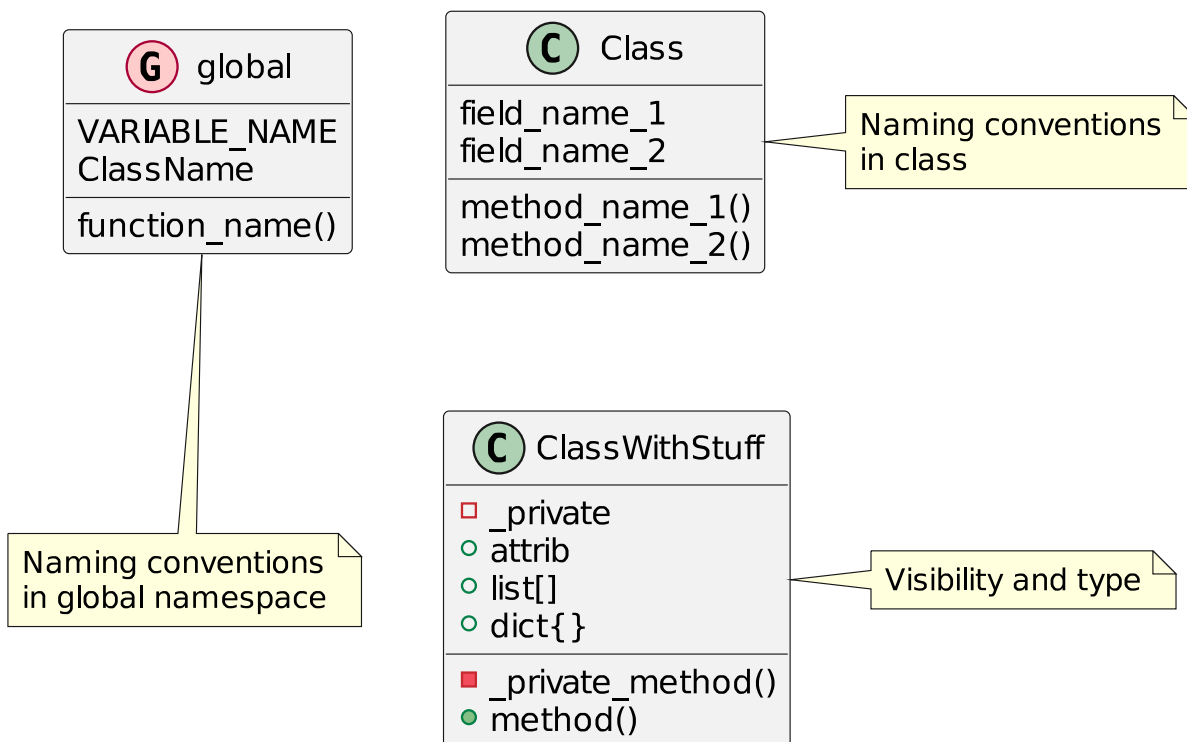
PlantUML offers various association types:

Relation	Symbol	Description
Association	--	no owner, navigability unspecified
Directed Association	<--	no owner, navigable
Association	x--	no owner, not navigable
Aggregation	o--	owner, but independent
Composition	*--	owner and dependent
Dependency	<..	no owner
Generalization/Extension	< --	
Realization	< ..	
Nesting	+--	

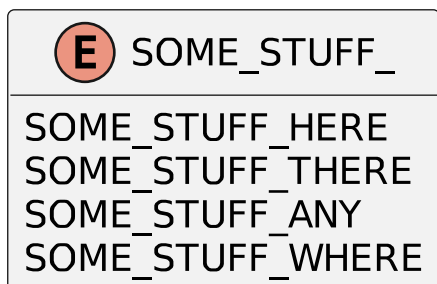
It is possible to replace -- by .. for broken lines.

Class Diagram Examples

Naming conventions, visibility and types in global namespace and within classes:



Enums can be used to describe variable collections:



Here is a more elaborate class diagram in `_static/big-brother.puml`:

```
@startuml
!include ws-cartoon-logo.puml
class "<$ws> BigBrother" as BB {
    who[]
    who_not[]
    --
    watch_em()    : who
    check_em()    : who_not
    ..
    check(p)      : suspect/vindicate
    watch(p,time) : boring/strange
    suspect(p)    : move //p// from //who_not// to //who//
    vindicate(p)  : move //p// from //who// to //who_not//
}
hide BB circle
@enduml
```

It can be included with the `uml` directive:

```
.. uml:: _static/big-brother.puml
:caption: Big Brother is watching you
```

or specify path to external [PlantUML](#) file:

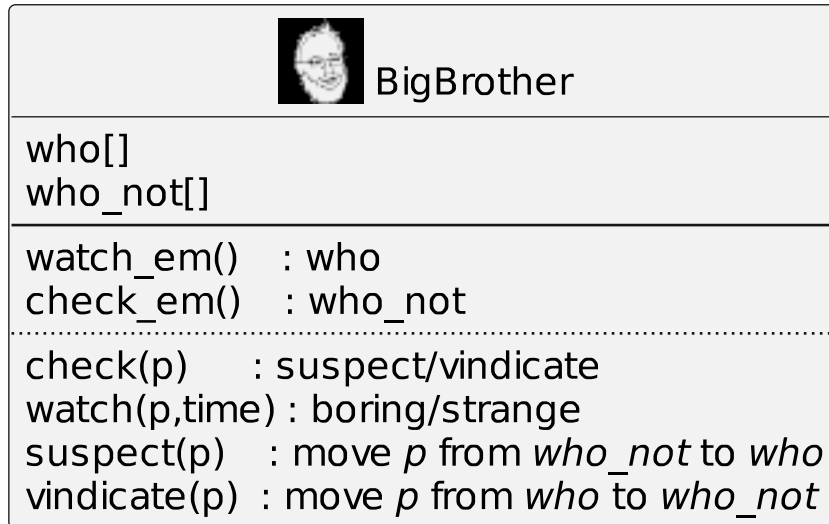
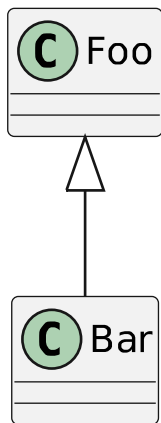
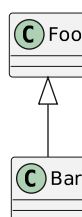


figure 12.1: Big Brother is watching you



You can specify height, width, scale and align:



You can specify a caption:

!todo: describe *uml* directive explicitly

If the *uml* directive has a caption (via option `:caption:`), it behaves like the *figure* directive. Otherwise, it behaves like the *image* directive.

12.2.3 Activity Diagram

Activity diagram is UML behavior diagram which shows flow of control or object flow with emphasis on the sequence and conditions of the flow. The actions coordinated by activity models can be initiated because other actions finish executing, because objects and data become available, or because some events external to the flow occur. (<https://www.uml-diagrams.org/activity-diagrams.html>)

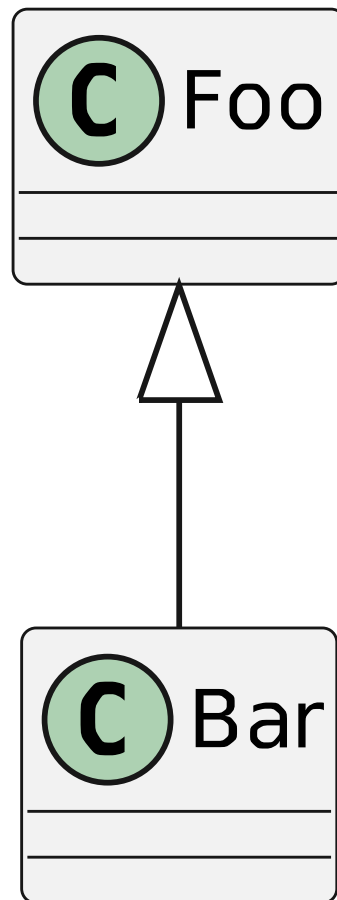


figure 12.2: Caption with **bold** and *italic*

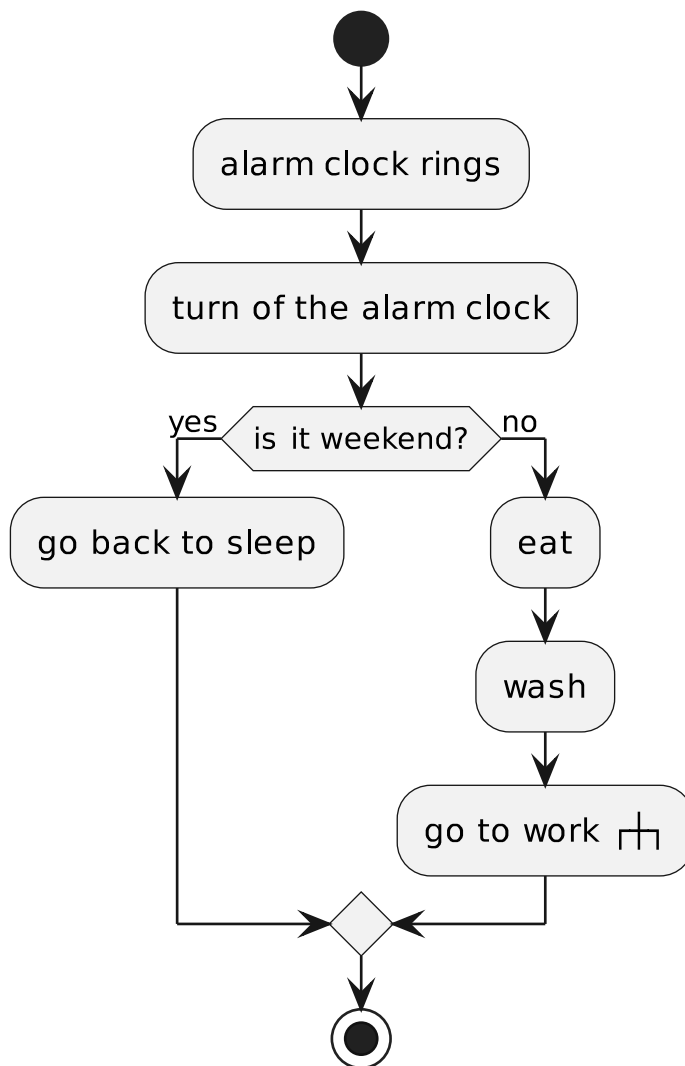
Note: It is **not** a data flow diagram!

See [State Machine Diagram vs Activity Diagram](#) for a basic explanation of the differences.

Here is the activity diagram of a persons daily routine:

```
.. uml::
    @startuml
    !include _static/call-bevahor.puml
    start
    :alarm clock rings;
    :turn off the alarm clock;
    if(is it weekend?) then (yes)
    :go back to sleep;
    else (no)
    :eat;
    :wash;
    :CALL(go to work);
    endif
    stop
    @enduml
```

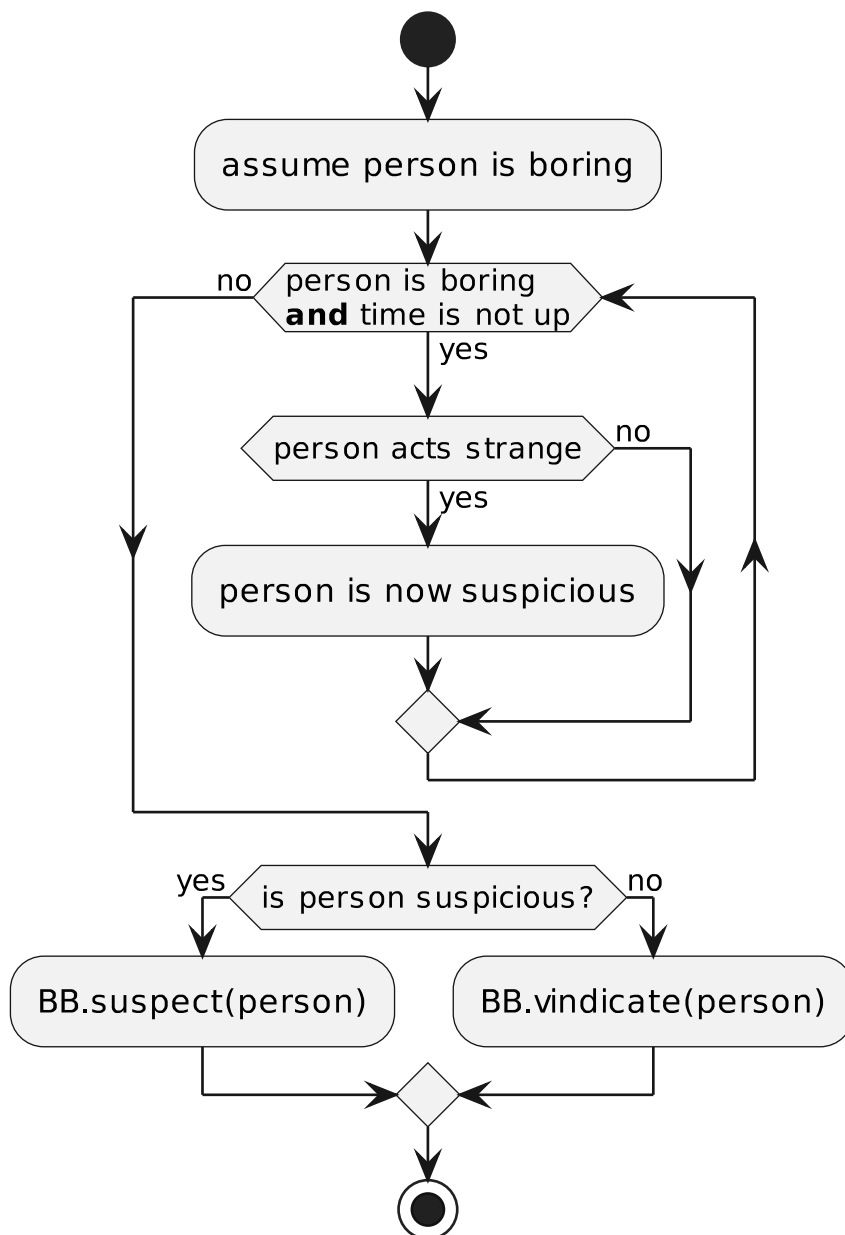
This is how it is rendered in Sphinx:



For details see [PlantUML Language Reference Guide](#).

Activity Diagram Examples

Here is the activity diagram to check a person *BB.check(person)*:



Another activity Diagram:

12.2.4 State Machine Diagram

State machine diagram is a behavior diagram which shows discrete behavior of a part of designed system through finite state transitions. State machine diagrams can also be used to express the usage protocol of part of a system. (<https://www.uml-diagrams.org/state-machine-diagrams.html>)

Here is the State Machine Diagram of the procedure of starting a software project:

```

.. uml::
  @startuml
  
```

(continues on next page)

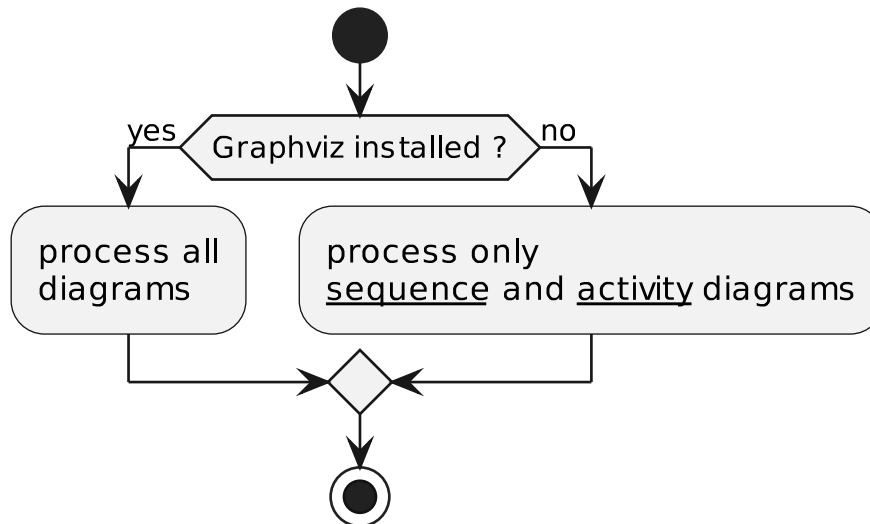


figure 12.3: Activity of the day

(continued from previous page)

```

state plan as "Planning Project"
plan : identify necessary UML diagram

state create as "Create UML Diagram"

state draw as "Draw UML Diagram"
draw : modify diagram

state bless as "Blessing"
bless : discuss UML Diagram with boss

state coding as "Programming"

state doc as "Document"

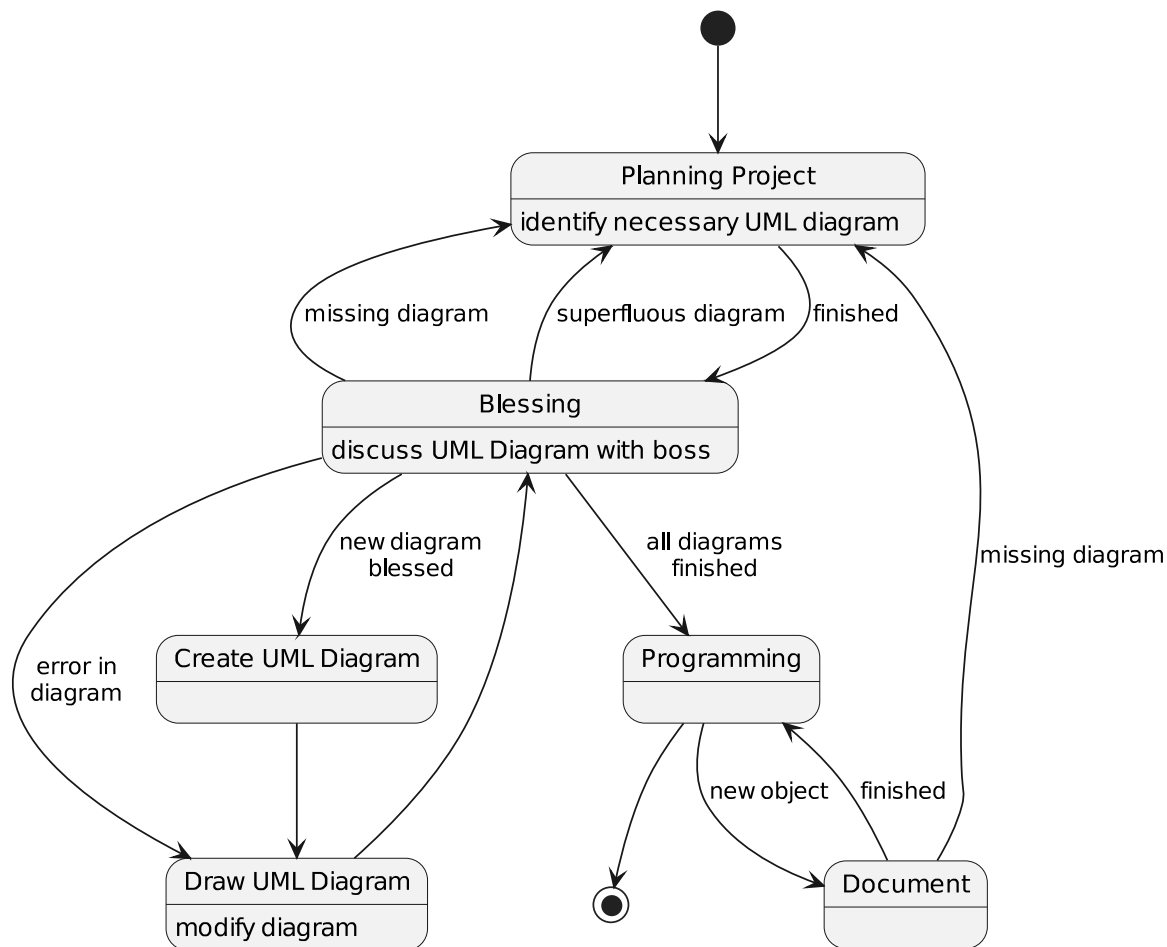
[*] --> plan
plan --> bless : finished
create --> draw
draw --> bless

bless --> plan : missing diagram
bless --> plan : superfluous diagram
bless --> create : new diagram\nblessed
bless --> draw : error in\ndiagram
bless --> coding : all diagrams\nfinished

coding --> doc : new object
doc --> coding : finished
doc --> plan : missing diagram
coding --> [*]
@enduml

```

This is how it is rendered in Sphinx:



See [Choice Pseudostate and Guard Condition in State Diagrams](#) for additional state diagram syntax:

```

@startuml
left to right direction

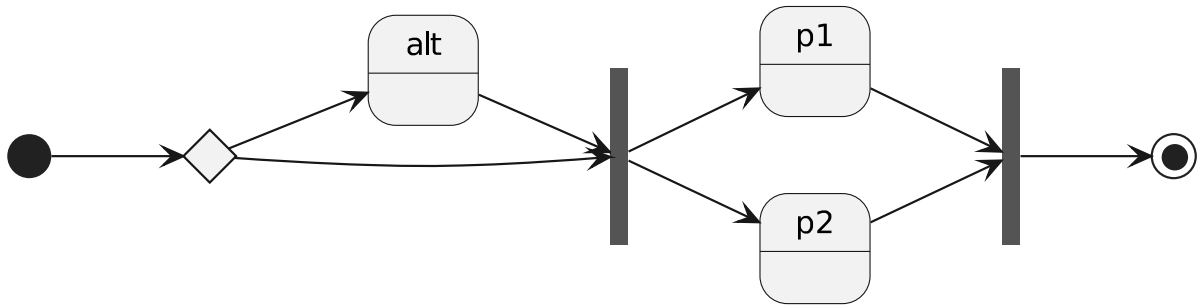
state choice <<choice>>
state alt
state fork <<fork>>
state p1
state p2
state join <<join>>
state end <<end>>

[*] --> choice
choice --> alt
alt --> fork

choice --> fork
fork --> p1
fork --> p2
p1 --> join
p2 --> join

join --> end
@enduml
    
```

rendered as:



12.2.5 Sequence Diagram

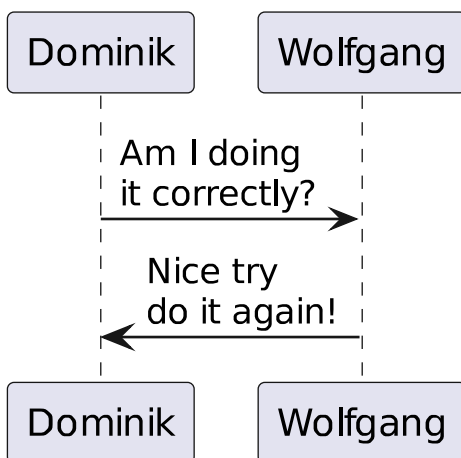
Sequence diagram is the most common kind of *interaction diagram*, which focuses on the message interchange between a number of lifelines.

Sequence diagram describes an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding occurrence specifications on the lifelines. (<https://www.uml-diagrams.org/sequence-diagrams.html>)

Here is the sequence diagram of a short conversation between me and my boss:

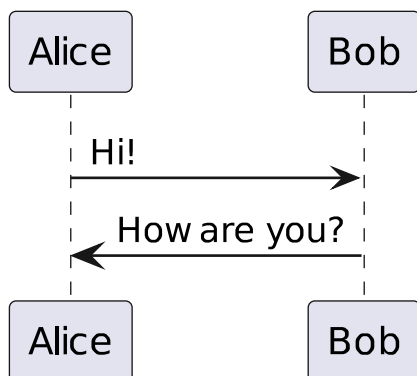
```
.. uml::
    Dominik -> Wolfgang : Am I doing\nit correctly?
    Wolfgang -> Dominik : Nice try\ndo it again!
```

This is how it is rendered in Sphinx:



For details see [PlantUML Language Reference Guide](#).

Sequence Diagram Examples



12.3 PlantUML

PlantUML is (in its own words) an

open-source tool that uses simple textual descriptions to draw UML diagrams.

See also sections *PlantUML Installation* and *PlantUML Emacs Mode* below.

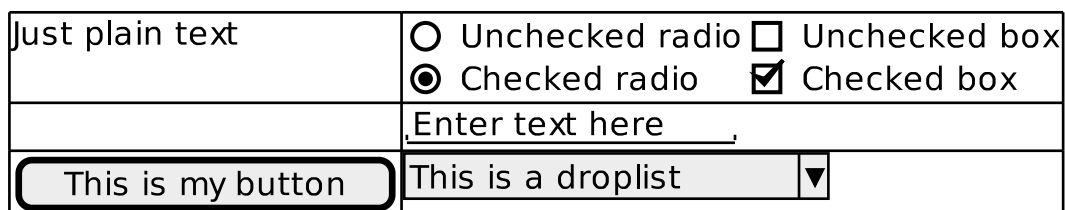
12.3.1 PlantUML Usage

The `PlantUML Language Reference Guide` is available locally and describes the standard `PlantUML` diagrams. However, some diagrams are only explained at the `PlantUML` web site.

Since `PlantUML` is the standard documentation tool for UML diagrams, specifying diagrams is explained, where the UML diagrams are described.

Here are just some experiments with special diagram types.

Salt is meant for GUI specifications:



12.3.2 PlantUML Installation

`PlantUML` is available as a ubuntu package:

```
apt-get install plantuml
```

Besides numerous other integrations there is also a `sphinx` extension module available:

```
apt-get install python-sphinxcontrib.plantuml
apt-get install python3-sphinxcontrib.plantuml
```

The standard ubuntu packages are too old (sphinx: 0.5)! Use the backported versions (sphinx: 0.11) which are available in the local repository:

```
deb [trusted=yes] http://scherer.wiedenmann.intern/repository xenial main
deb-src [trusted=yes] http://scherer.wiedenmann.intern/repository xenial main
```

12.3.3 PlantUML Emacs Mode

There is also a major mode for editing PlantUML sources in Emacs with preview facilities:

```
wget https://raw.githubusercontent.com/skuro/plantuml-mode/master/plantuml-mode.el
```

This mode is included in the local shared site-lisp.

Enable the Major Mode

You can automatically enable `plantuml-mode` for files with extension `.plantuml` by adding the following to your `.emacsrc`:

```
;; Enable plantuml-mode for `PlantUML` files
(add-to-list 'auto-mode-alist '("\\.plantuml\\") . plantuml-mode))
```

Of course, you can always enable manually the major mode by typing `M-x plantuml-mode` once in the desired PlantUML file buffer.

Emacs Mode Usage

You can tell `plantuml-mode` to autocomplete the word before the cursor by typing `M-x plantuml-complete-symbol`. This will open a popup with all the available completions as found in the list of keywords given by running PlantUML with the `-language` flag.

To render the PlantUML diagram within Emacs, you can hit `M-x plantuml-preview`. This will run `plantuml(1)` and display the result in the `*PLANTUML-Preview*` buffer. The format used to render the diagram is automatically chosen from what's supported by your Emacs. It will be one of the following, in order of preference:

- SVG
- PNG
- Unicode ASCII art

The diagram will be either created from the selected region if one is available in the current buffer, or using the whole buffer otherwise.

If you want to force a specific output format, you can customize the variable `plantuml-output-type` to the value you prefer.

Default Key Bindings

The following shortcuts are enabled by default:

```
C-c C-c plantuml-preview: renders a PlantUML diagram from the current buffer in the best
↳supported format
C-u C-c C-c plantuml-preview in other window
C-u C-u C-c C-c plantuml-preview in other frame
```

UMLX

The python module *UML annotations - line_diversion* extracts marked annotation lines from source files.

Emacs Mode Installation

To enable preview you need to tell `plantuml-mode` where to locate the `PlantUML` JAR file. By default it will look for it in `~/plantuml.jar`, but you can specify any location with:

```
M-x customize-variable<RET>
plantuml-jar-path<RET>
```

12.4 yUML

yUML is a web service only, which makes it hard to use offline.

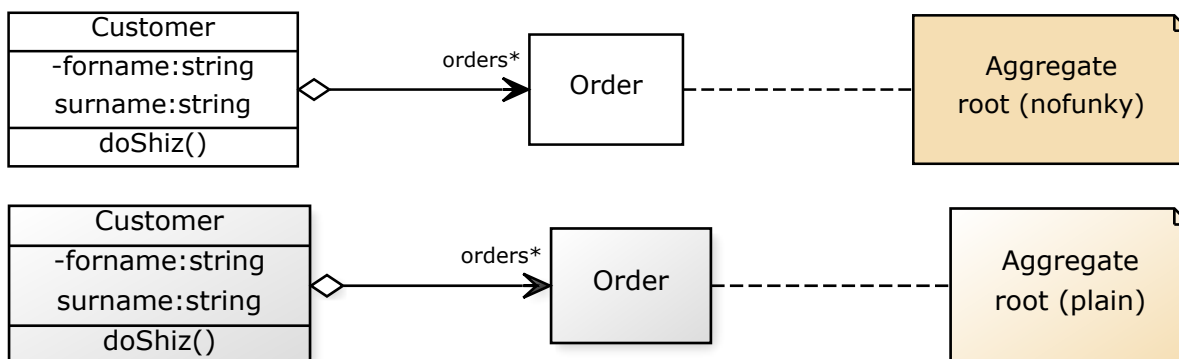
There is an [unofficial command line tool for yuml and sphinx integration sphinxcontrib-yuml](#).

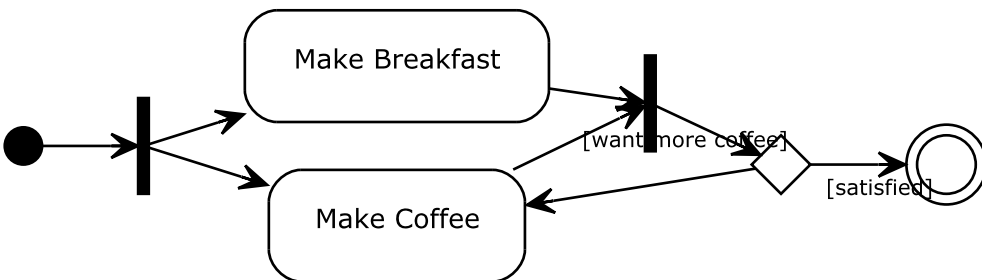
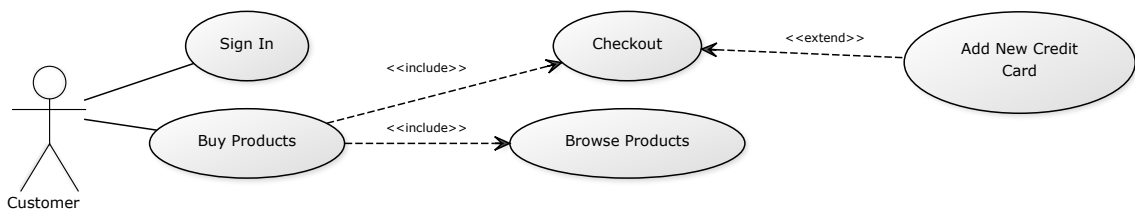
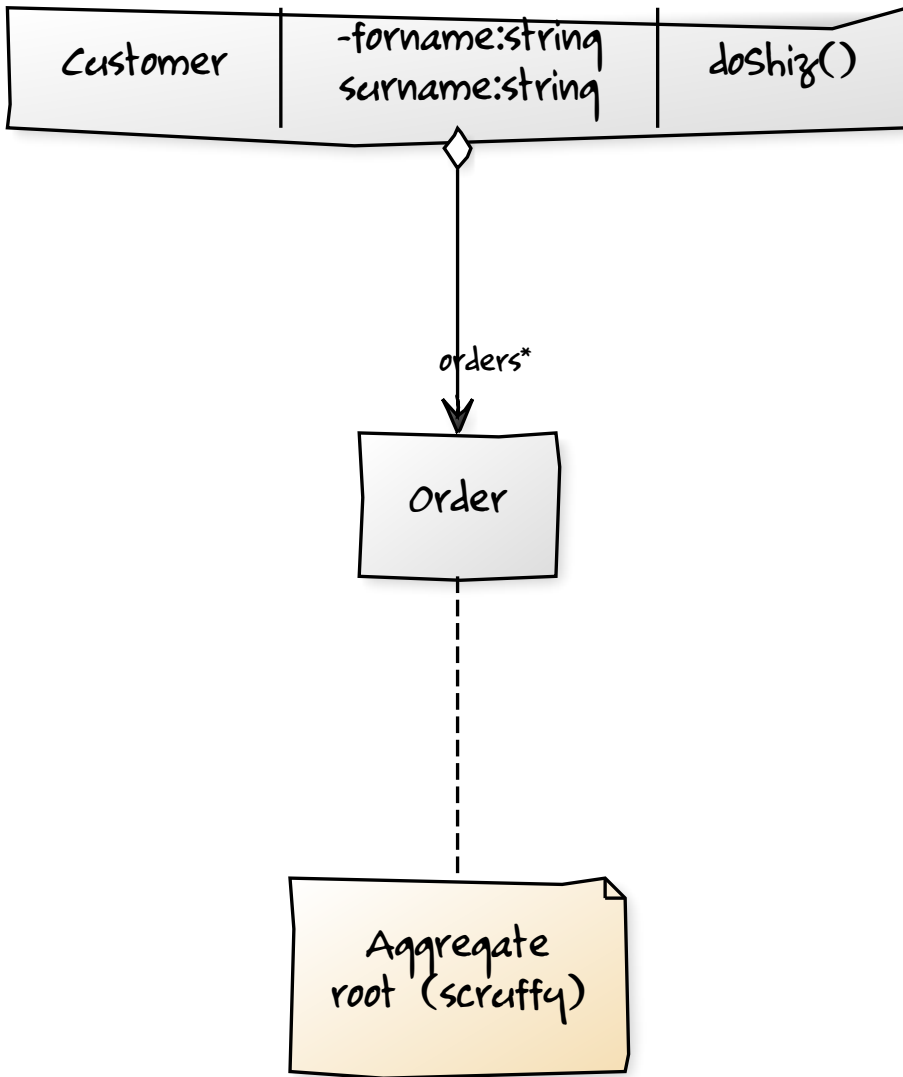
[GitHub - aivarsk/scruffy](#) is an abandoned project that does not use the web service.

yUML provides support for class, activity and usecase diagrams:

```
.. yuml::
  :alt: [Customer]->[Orders]
  :type: class, activity or usecase
  :scale: positive integer value
  :direction: LR, TD or RL (ignored by usecase and activity)
  :style: nofunky, plain, scruffy

  [Customer|-forname:string;surname:string|doShiz()]<->-orders*>[Order]
  [Order]-[note:Aggregate root (nofunky){bg:wheat}]
```





12.4.1 Installation

yUML command line tool:

```
easy_install git://github.com/wandernauta/yuml
```

The updated package `python-sphinxcontrib.yuml` is in the local package repository and can be installed with:

```
apt-get install python-sphinxcontrib.yuml
```

See also [njouanin/sphinxcontrib-yuml](#);

Manual installation from original sources:

```
wget -q 'https://pypi.python.org/packages/bf/9b/
↪99d1ea03b6199ccd93fbf19b02fe23160b0004b6973d4bd1ef233bd633e2/sphinxcontrib-yuml-0.3.1.tar.gz
↪'
tar -zxf sphinxcontrib-yuml-0.3.1.tar.gz
(
cd sphinxcontrib-yuml-0.3.1/sphinxcontrib/ || exit 1
patch -p0 <<'EOF'
diff -u yuml.orig.py yuml.py
--- yuml.orig.py      2013-11-19 22:08:16.000000000 +0100
+++ yuml.py 2018-04-09 10:47:29.740133938 +0200
@@ -15,6 +15,7 @@
     from os import path
     from docutils import nodes
     from docutils.parsers.rst import directives
+import docutils.parsers.rst.directives.images
     from sphinx.errors import SphinxError
     from sphinx.util import ensuredir, relative_uri
     try:
@@ -48,13 +49,13 @@
         :type: class, activity or usecase
         :scale: positive integer value
         :direction: LR, TD or RL
-        :style: boring, plain, scruffy
+        :style: nofunky, plain, scruffy

         [Customer]->[Billing Address]
         """
         type_values = ('class', 'activity', 'usecase')
         direction_values = ('LR', 'RL', 'TD')
-        style_values=('boring', 'plain', 'scruffy')
+        style_values=('nofunky', 'plain', 'scruffy')

         def type_choice(argument):
             return directives.choice(argument, YumlDirective.type_values)
EOF
cp yuml.py /usr/lib/python2.7/dist-packages/sphinxcontrib/
)
```

12.5 Other UML Tools

Other interesting UML tools by category.

12.5.1 Declarative

- yUML is a web service.
- UMLGraph

Class diagrams: Java syntax, Javadoc integration, `umlgraph` -> `dot` -> image.

Sequence diagrams: `pic` macros -> `pic2plot` -> image.

12.5.2 GUI with UML standard support

According to the list of Unified Modeling Language tools on Wikipedia, The most recently active projects are

- **UML Designer** (Eclipse) best handling
- **Papyrus** (Eclipse)
- **BOUML** - a free UML tool box has a python generator (free of use)
- **Modelio Open Source** - UML and BPMN free modeling tool (no SVG?)
- **Umbrello** (package **Umbrello**)
UML standard support, XMI file format, image export.
- **UMLet** (package **umlet**)
Freeform diagrams, proprietary XML file format, image export. Java auto generator support.
- **ArgoUML** (UML 1.4)
UML standard support, XMI file format, image export.

12.5.3 Generic Diagrams

- **Dia** (package **dia**),
Freeform diagrams, proprietary compressed XML format. Image, **dot**, Visio export.

12.5.4 Auto generators

- **gen_plantuml.py** based on the GitHubGist generate PlantUML definition from python sources.
- **pyreverse** (from package **pylint**),
python -> dot.

Generating a PlantUML definition of global variables and functions can also be as simple as:

```

1 class_name = 'global'
2
3 func_type = type(run)
4 func_type_builtin = type(issequence)
5 module_type = type(os)
6
7 globals_ = globals()
8 public = globals_.get('__all__', [])
9 if not public:
10     public = list(dkeys(globals_))
11
12 items = []
13 items.extend(((_sym, _val, '-') for _sym, _val in sorted(ditems(globals_)) if _sym not in_
↪public))
14 items.extend(((_sym, globals_[_sym], '+') for _sym in public))
15
16 opt_all_syms = None
17
18 classes = []
19 variables = []
20 functions = []
21 for _sym, _val, _visibility in items:
22     if not opt_all_syms:
23         if _sym.startswith('_'):
24             continue
25         if _sym.startswith('__') and _sym.endswith('__'):
26             continue
27         if isinstance(_val, module_type):
28             continue

```

(continues on next page)

(continued from previous page)

```

29     if isinstance(_val, type):
30         classes.append((_sym, _val, _visibility))
31         continue
32
33     if isinstance(_val, (func_type, func_type_builtin)):
34         _list = functions
35         _sfx = '()'
36     else:
37         _list = variables
38         _sfx = ''
39     _list.append(sformat('{0} : {1}{2}{3}', class_name, _visibility, _sym, _sfx))
40
41 printf(sformat('    class {0} << (G, #FFCCCC) >>', class_name))
42 for _list in variables, functions:
43     printf('\n'.join('        ' + _e for _e in _list))

```

Or as fancy as:

```

1  PARENTHESES = dict((
2      ('(', ')'),
3      ('[', ']'),
4      ('{', '}'),
5      ('<', '>'),
6      ('<<', '>>'),
7      ('|', '|'),
8      (' ', ' '),
9      ('"', '"'),
10     ('"', '"'),
11 ))
12
13 __all__.append('enclose')
14 def enclose(elt, open=None, close=None, sep=None, forced=None): # ||:fnc:||
15     r"""enclose string in parenthesis.
16
17     :returns: enclosed string
18
19     :param elt:     element to be enclosed
20     :param open:   opening parenthesis (default: `(`)
21     :param close:  closing parenthesis (default: `)`PARENTHESES.get(open, '')
22     :param sep:    separator after open and before close, (default: '')
23     :param forced: add parentheses, even if they already exist.
24
25     >>> printf(enclose('abc'))
26     (abc)
27
28     >>> printf(enclose('abc'))
29     (abc)
30
31     >>> printf(enclose('abc'))
32     (abc)
33
34     >>> printf(enclose('(abc)'))
35     (abc)
36
37     >>> printf(enclose('(abc)', forced=True))
38     ((abc))
39
40     >>> printf(enclose('abc', '<<'))
41     <<abc>>
42
43     >>> printf(enclose(enclose('abc', ' '), '<<'))
44     << abc >>
45
46     >>> printf(enclose('abc', '-| '))
47     -| abc
48
49     """
50
51     if open is None:
52         open = '('

```

(continues on next page)

(continued from previous page)

```

53     if close is None:
54         close = PARENTHESSES.get(open, '')
55     if sep is None:
56         sep = ''
57
58     enclosed = []
59     if forced or (open and not re.match('\s*' + re.escape(open), elt)):
60         enclosed.append(open)
61         if open:
62             enclosed.append(sep)
63
64     enclosed.append(elt)
65     if forced or (close and not re.search(re.escape(close) + '\s*$', elt)):
66         if close:
67             enclosed.append(sep)
68             enclosed.append(close)
69     return ''.join(enclosed)
70
71 __all__.append('join_lists_flatten')
72 def join_lists_flatten(lists, sep): # ||:fnc:||
73     r"""Insert separator between elements of a list.
74
75     :returns: list where each group of elements is separated from the next
76             group of element with `sep`.
77
78     :param lists: a list of element groups (lists). All elements of
79                 group are appended as single elements to the result list.
80     :param sep: separator inserted between two groups of elements.
81
82     """
83     result = []
84     if lists:
85         result.extend(lists[0])
86     for _list in lists[1:]:
87         result.append(sep)
88         result.extend(_list)
89     return result
90
91 __all__.append('plantuml_format_class')
92 def plantuml_format_class(name, sections=None, circle=None, stereotype=None, alias=None,
93 ↪ sep=None): # ||:fnc:||
94     r"""
95     :returns: class formatted for a `PlantUML`_ class diagram.
96
97     :param name: class name
98     :param sections: list of class sections, each section is a list of attribute/operation
99 ↪ lines.
100     :param circle:
101     :param stereotype:
102     :param alias: class alias
103
104     .. _`PlantUML`: http://plantuml.com
105     """
106     if sections is None:
107         sections = []
108
109     formatted = []
110
111     _class_param = []
112     if circle and circle.strip():
113         _class_param.append(enclose(circle, '()'))
114     if stereotype and stereotype.strip():
115         _class_param.append(stereotype)
116     class_param = ' '.join(_class_param)
117     if class_param:
118         class_param = enclose(class_param, '<<', sep=' ')
119
120     if alias:
121         name = enclose(name, '')
122         class_alias = 'as ' + alias

```

(continues on next page)

```

122     else:
123         class_alias = ''
124
125     if sep is None:
126         sep = ''
127
128     formatted.append(sformat('class {0} {{{', ' '.join((_s for _s in (name, class_alias, class_
↳param) if _s))))
129
130     _first = True
131     for _list in sections:
132         if _list:
133             if _first:
134                 _first = False
135             else:
136                 formatted.append(' ' + sep)
137                 formatted.extend((' ' + _e for _e in _list))
138
139     formatted.append('}')
140     return '\n'.join(formatted)
141
142 class_name = 'global'
143
144 func_type = type(run)
145 func_type_builtin = type(issequence)
146 module_type = type(os)
147
148 globals_ = globals()
149 public = globals_.get('__all__', [])
150 if not public:
151     public = list(dkeys(globals_))
152
153 items = []
154 items.extend(((_sym, _val, '-') for _sym, _val in sorted(ditems(globals_)) if _sym not in_
↳public))
155 items.extend(((_sym, globals_[_sym], '+') for _sym in public))
156
157 opt_all_syms = None
158
159 import pyjsmo
160 classes = []
161 variables = pyjsmo.OrderedDict((
162     ('-', []),
163     ('+', []),
164 ))
165 functions = pyjsmo.OrderedDict((
166     ('-', []),
167     ('+', []),
168 ))
169 for _sym, _val, _visibility in items:
170
171     if not opt_all_syms:
172         if _sym.startswith('_'):
173             continue
174         if _sym.startswith('__') and _sym.endswith('__'):
175             continue
176         if isinstance(_val, module_type):
177             continue
178
179     if isinstance(_val, type):
180         classes.append((_sym, _val, _visibility))
181         continue
182
183     if isinstance(_val, (func_type, func_type_builtin)):
184         _list = functions[_visibility]
185         _sfx = '()'
186     else:
187         _list = variables[_visibility]
188         _sfx = ''
189     _list.append(sformat('{0}{1}{2}', _visibility, _sym, _sfx))
190

```

(continues on next page)

(continued from previous page)

```

191 variables = join_lists_flatten([_v for _v in dvalues(variables) if _v], '..')
192 functions = join_lists_flatten([_v for _v in dvalues(functions) if _v], '..')
193
194 printf(plantuml_format_class(class_name, (variables, functions), circle='G, #FFCCCC', sep='--
↳', alias='G'))

```

12.6 Summary

- UML is the abbreviation of unified modeling language. It is a standardized method for describing programs.
- The UML diagrams are easy to understand even for nonprogrammers
- There are existing some UML GUI-TOOLS. There are good for nonprogrammers to modeling something, but programmers shouldn't use them, because the not-diagram-output isn't readable for a human. And if the output is not readable, it needs more effort to write or update a documentation (you would need every time this GUI-Tool if you want to know what happens in the program via UML, without the GUI-TOOL, the output is worthless)
- Programmers should use [PlantUML](#) to create UML diagrams.
 - [PlantUML](#) is an open source tool, which uses a certain syntax to create a UML diagram
 - Programmers don't need the diagram output, to get an accurate overview of the program, because it is quite easy to understand the associations of the program if you look at the [PlantUML](#) syntax
 - [PlantUML](#) is already integrated in a lot of tools (for instance emacs, sphinx), so you can write your [PlantUML](#) code in for example in your python script as a command, and emacs can show you the diagram with a shortcut.
- A class diagram (similar to entity relationship diagrams for databases) shows boxes for class, interface, annotation, enum, ...

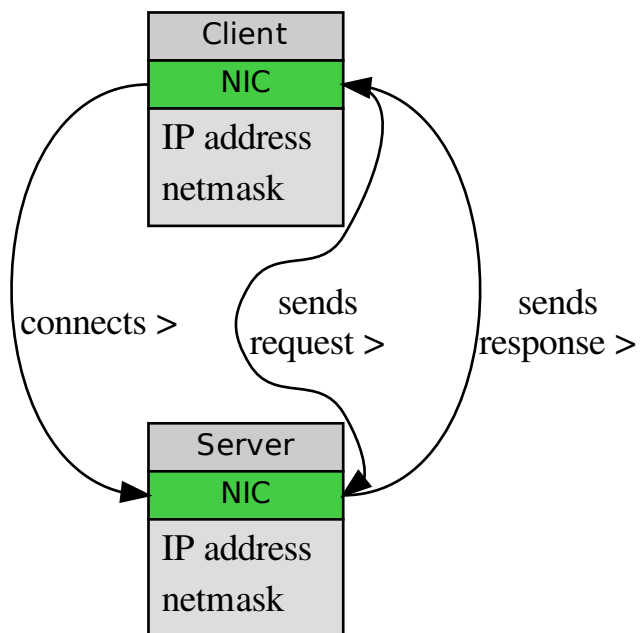
The upper section shows attributes, the lower section contains operations (methods).

- every class in its own box, globals in a own box
- different types of interfaces:
 - * Association – is independent (no owner) (students, teacher)
 - * Aggregation o– independent, no owner (department, teacher)
 - * Composition *__ death relationship, if parent dies, all children will die too (kettle and tableau)
- Sequence Diagram

Shows the message flow between several participants of a system on timelines.
- Activity Diagram (in German known as “Aktivitätsdiagramm”)

12.6.1 Object diagram with dot(1)

Why dot(1) is not so useful for object diagrams.



Besides the scripts from [section 2.5, Document administration](#), there are a couple of local tools available.

The script `bin/inst.sh` installs the programs in the appropriate places. See [section Activity Diagrams for bin/inst.sh](#).

13.1 Document Generation Issues

- ImageMagick permissions problems with PDF (see [imagemagick - convert:not authorized aaaa @ error/constitute.c/ReadImage/453](#)):

In file `/etc/ImageMagick-6/policy.xml` (or `/etc/ImageMagick/policy.xml`)

1. comment line (security risk!)

```
<!-- <policy domain="coder" rights="none" pattern="MVG" /> -->
```

2. change lines

```
<policy domain="coder" rights="none" pattern="PS" />
<policy domain="coder" rights="none" pattern="EPS" />
<policy domain="coder" rights="none" pattern="PDF" />
```

to

```
<policy domain="coder" rights="read|write" pattern="PS" />
<policy domain="coder" rights="read|write" pattern="EPS" />
<policy domain="coder" rights="read|write" pattern="PDF" />
```

3. add line

```
<policy domain="coder" rights="read|write" pattern="LABEL" />
```

- ImageMagick conversion problems SVG -> PNG with inkscape (see [convert aus ImageMagick verhält sich anders](#))

In file `/etc/ImageMagick-6/delegates.xml` (or `/etc/ImageMagick/delegates.xml`)

1. change line

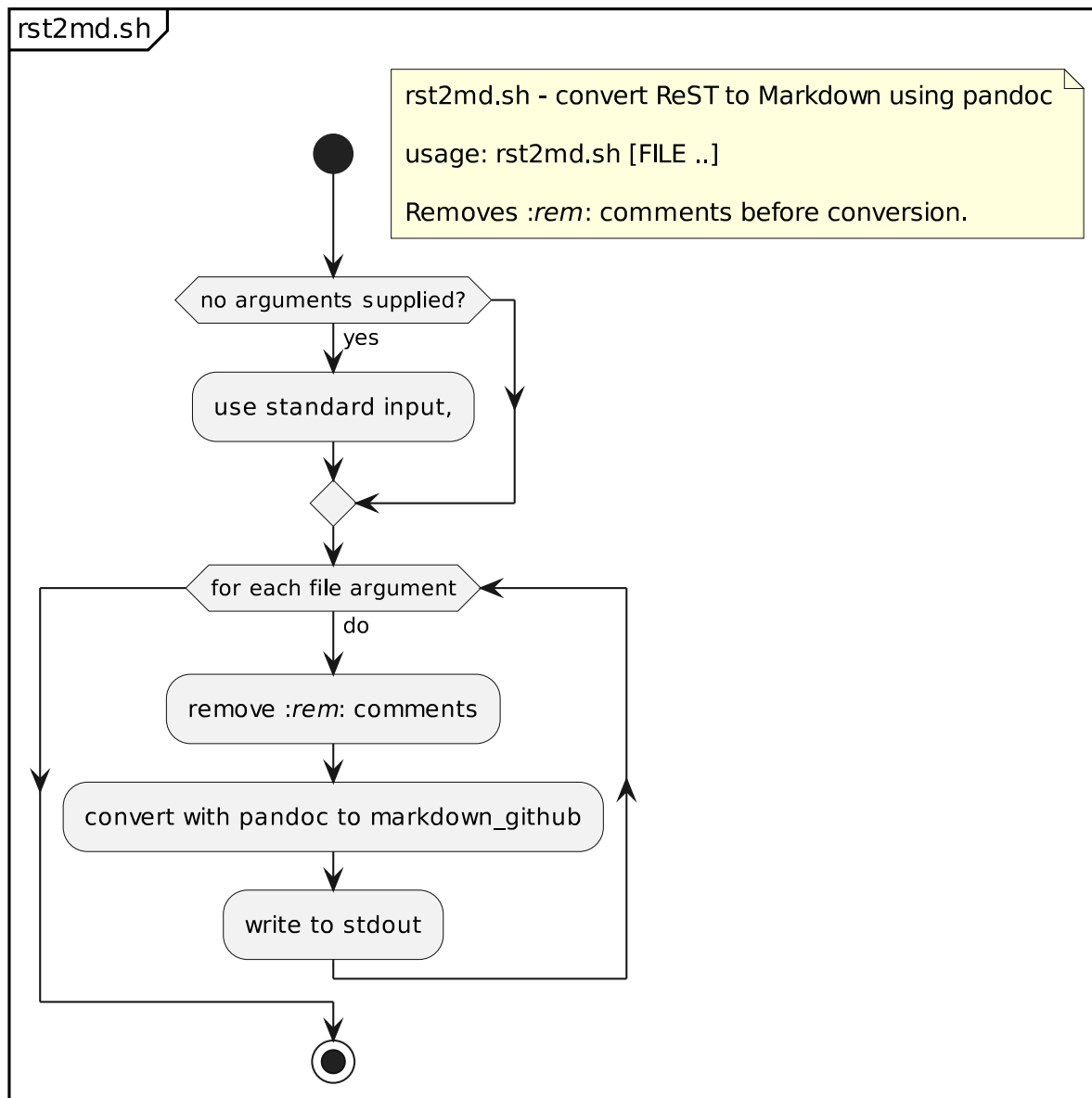
```
<delegate decode="svg:decode" stealth="True" command="&quot;inkscape&quot; &quot;%s&
↳ &quot; --export-eps=&quot;%s&quot; --export-dpi=&quot;%s&quot; --export-
↳ background=&quot;%s&quot; --export-background-opacity=&quot;%s&quot; &gt; &quot;
↳ %s&quot; 2&gt;&amp;1"/>
```

to

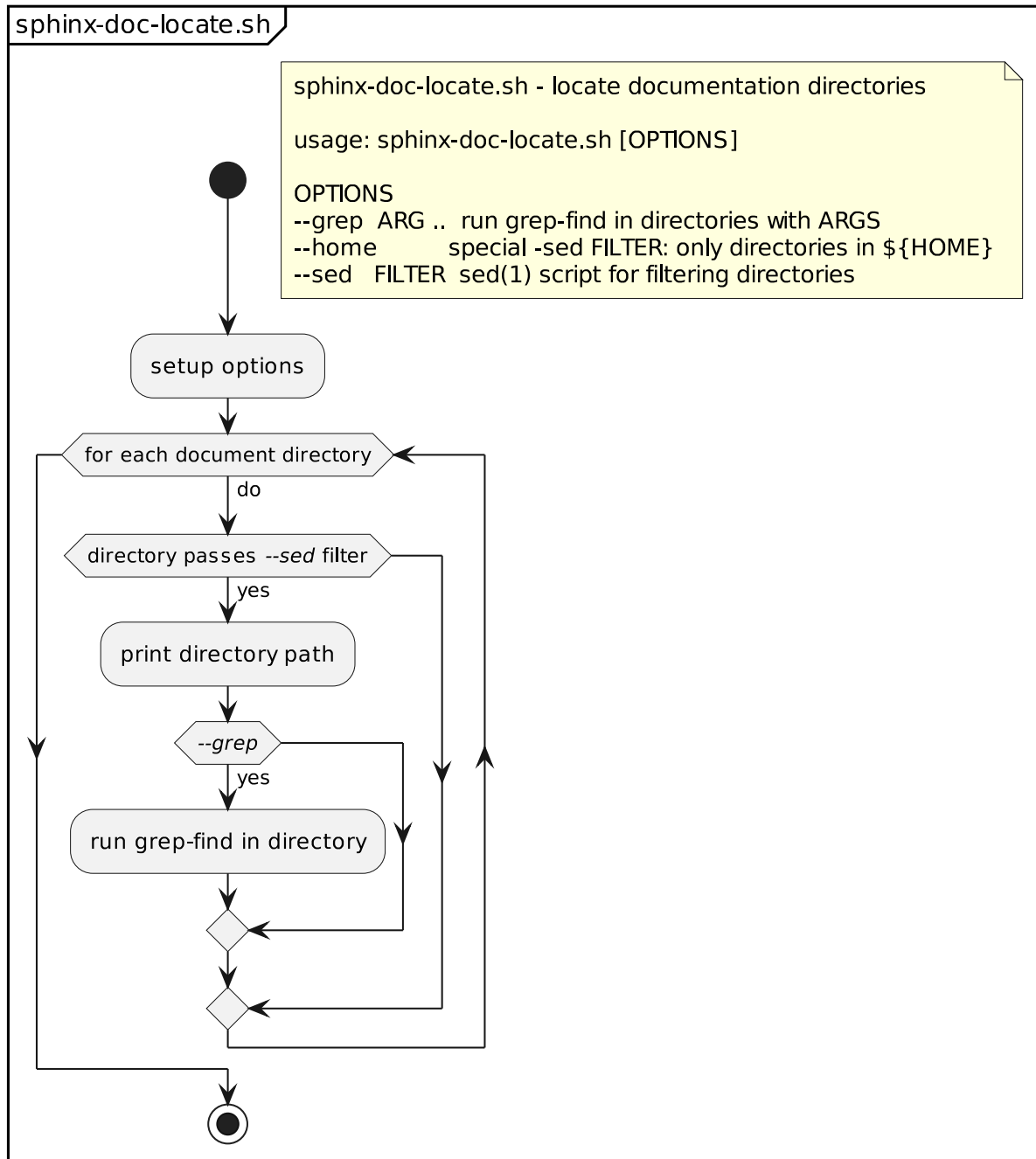
```
<delegate decode="svg:decode" stealth="True" command="&quot;inkscape&quot; &quot;%s&
↳ &quot; --export-png=&quot;%s&quot; --export-dpi=&quot;%s&quot; --export-
↳ background=&quot;%s&quot; --export-background-opacity=&quot;%s&quot; &gt; &quot;
↳ %s&quot; 2&gt;&amp;1"/>
```

This avoids the mentioned artifacts (actually caused by ghostscript, not inkscape).

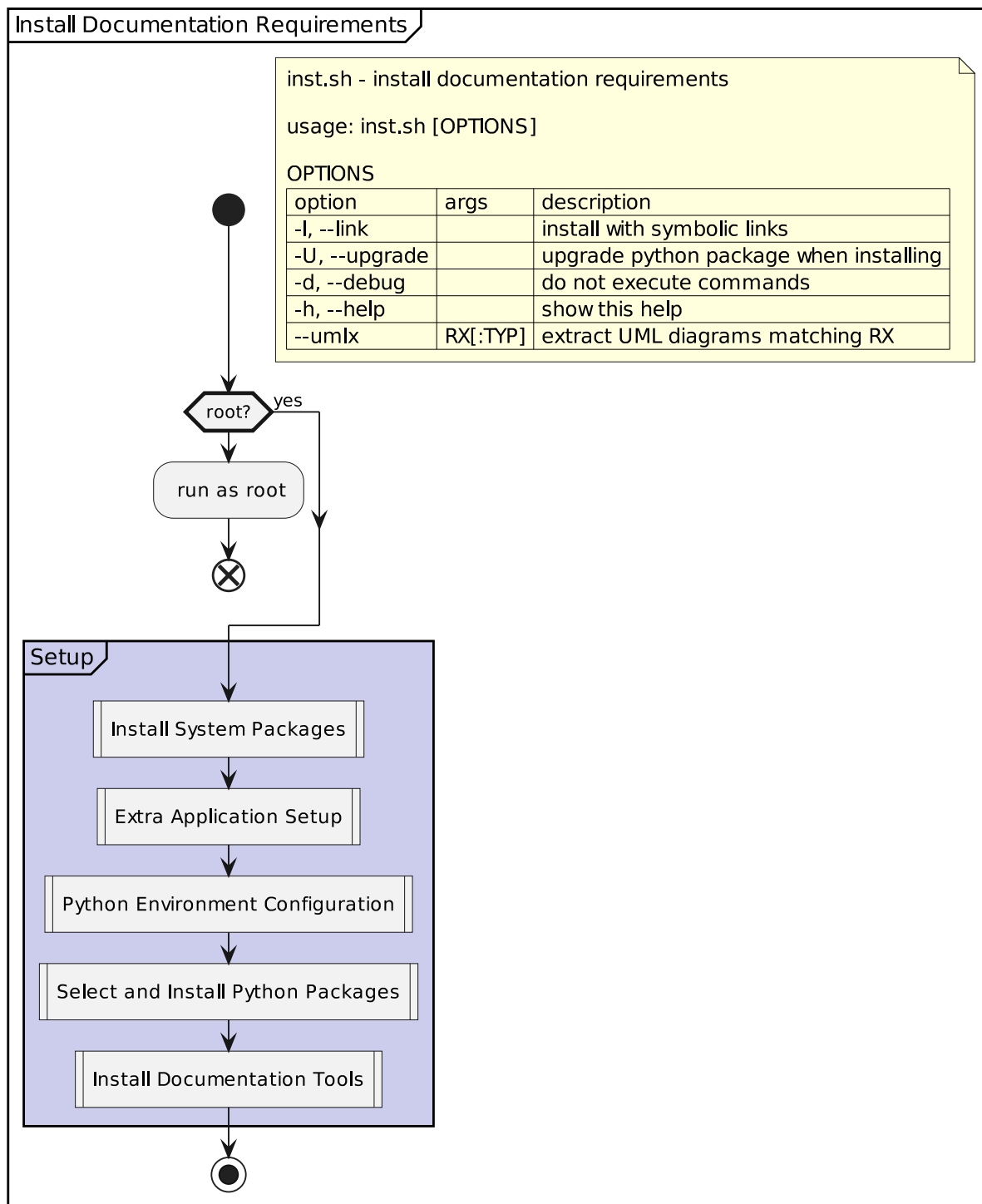
13.2 Activity Diagrams for `rst2md.sh`

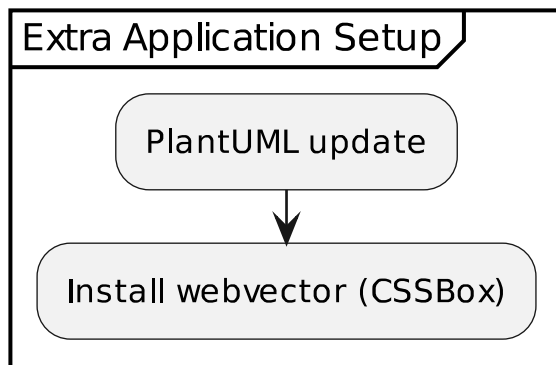
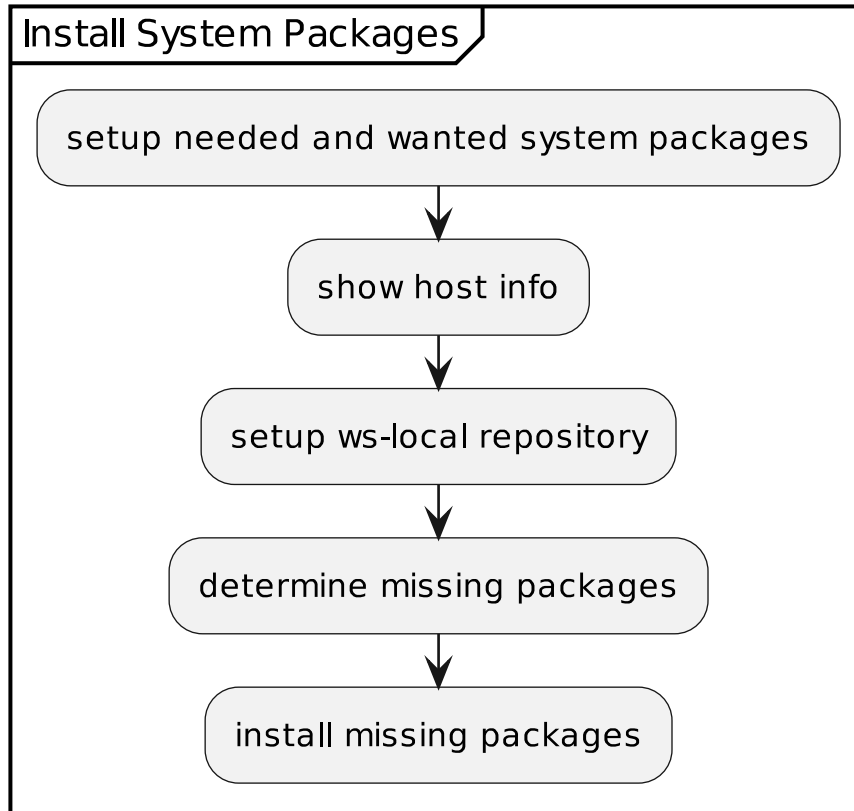


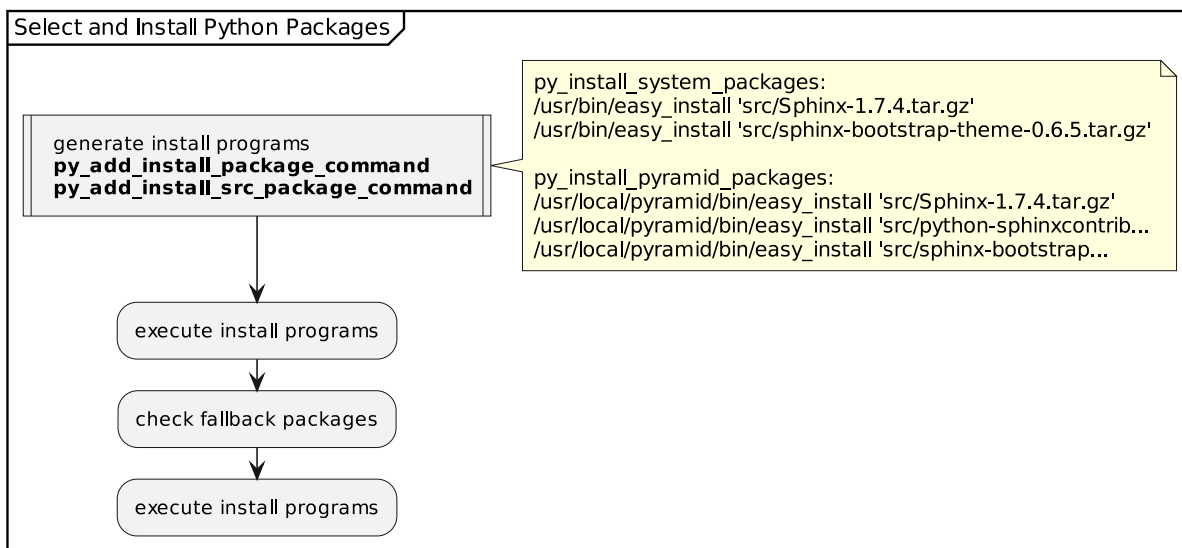
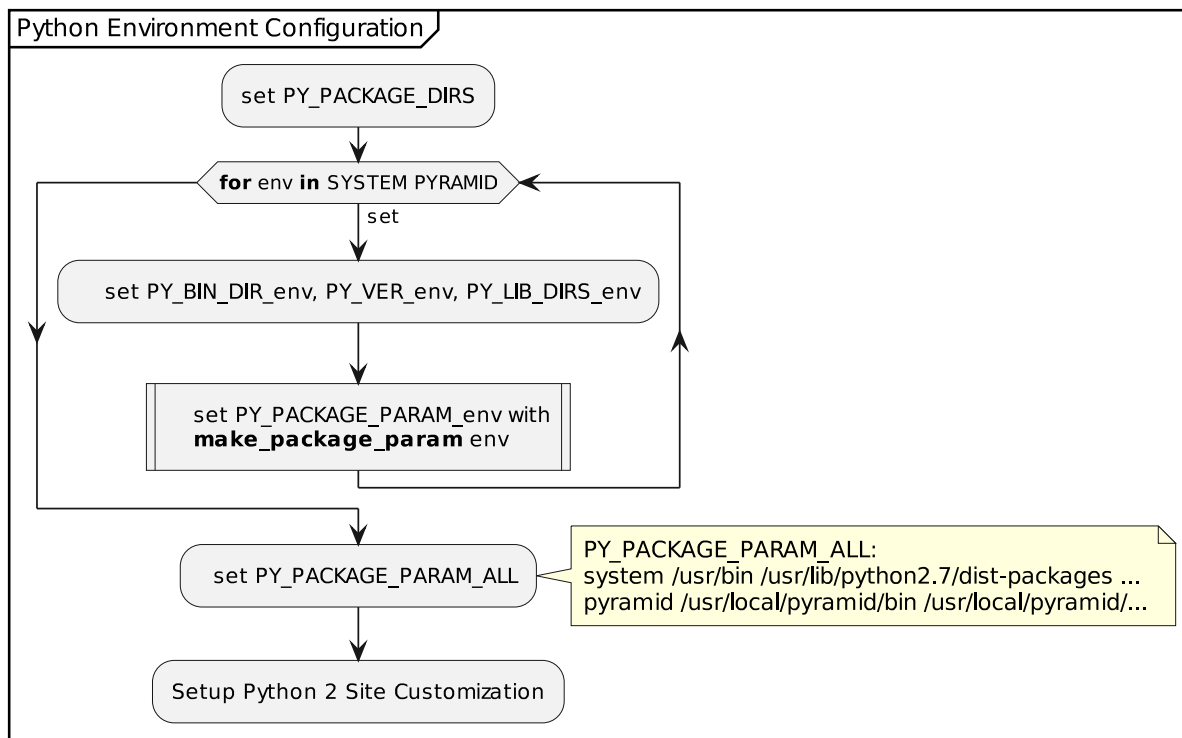
13.3 Activity Diagrams for sphinx-doc-locate.sh



13.4 Activity Diagrams for bin/inst.sh







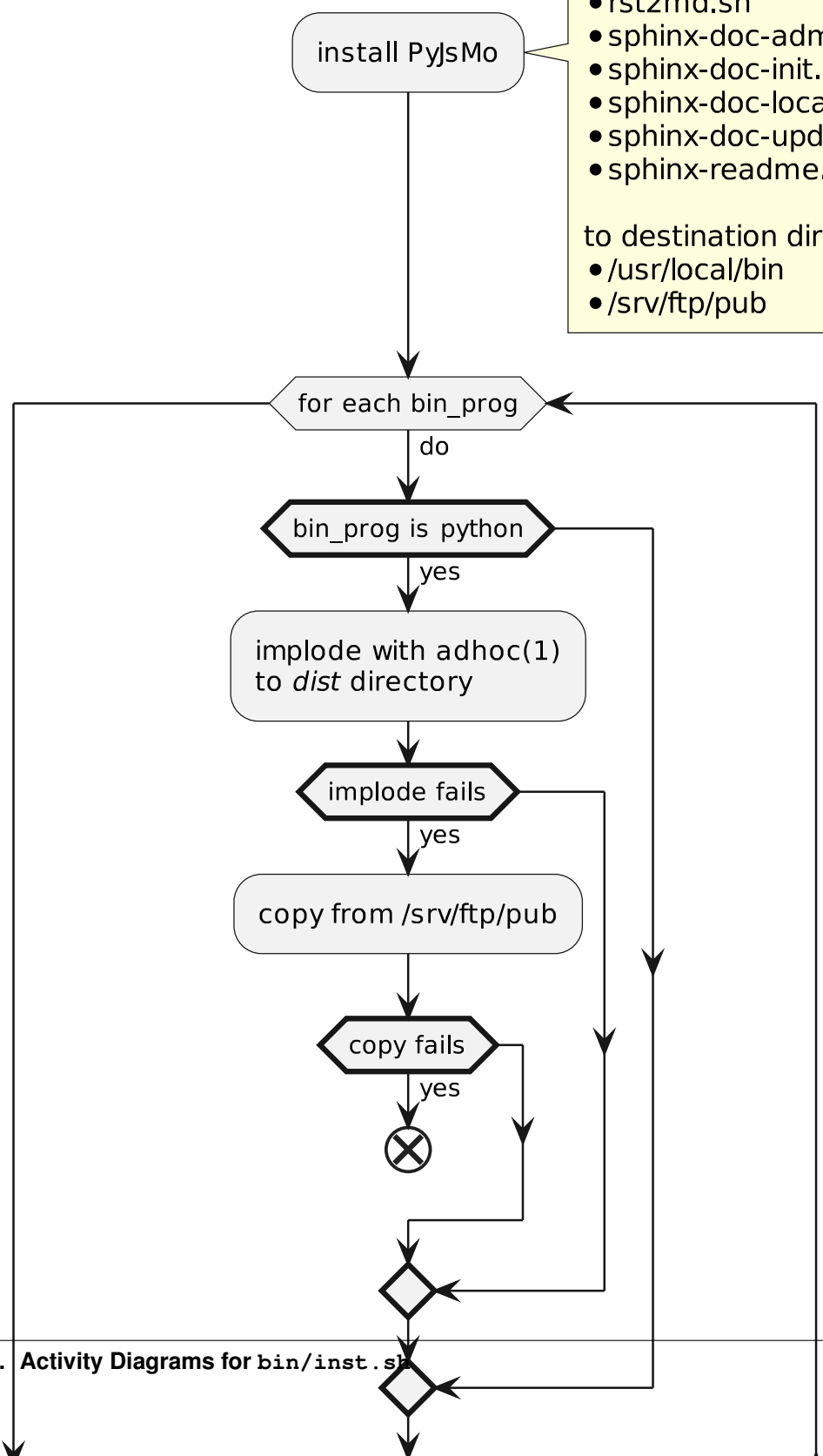
Install Documentation Tools

install bin_progs:

- find_documentation.py
- grep-doc.sh
- line_diversion.py
- sphinx_doc_snip.py
- sphinx_doc_glossary.py
- rst2md.sh
- sphinx-doc-admin.sh
- sphinx-doc-init.sh
- sphinx-doc-locate.sh
- sphinx-doc-update.sh
- sphinx-readme.sh

to destination directories:

- /usr/local/bin
- /srv/ftp/pub



SPHINX DOCUMENTATION GENERATOR

The basic requirements for a documentation generator are as follows:

- Must extract documentation from source code. ([Sphinx](#), [Doxygen](#)).
- Must include full source code with the documentation. ([Sphinx](#), [Doxygen](#)).
- Should be extensible for integration of format converters.
[Sphinx](#) extensions are simple, as it is written in Python.
[Doxygen](#) extensions are virtually non-existent, since [Doxygen](#) is written in C++. See the [PlantUML integration with Doxygen](#) for an elaborate example.
- Should be hackable. ([Sphinx](#)).
- Should come with source code. ([Sphinx](#), [Doxygen](#)).

Since [Sphinx](#) is easily extendable, because there is a [Doxygen](#) to [Sphinx](#) bridge, and it is ultimately hackable, [Sphinx](#) is the logical choice for overall documentation.

14.1 ReStructuredText Tips and Issues

14.1.1 Spaces at the beginning of formatted text

There is a problem with blank spaces at the beginning of formatted text. It cannot be solved with no-break spaces (C-q 240 RET).

E.g., this:

```
| NO-BREAK SPACE: >` ` `<  
| NO-BREAK SPACE: >:literal:` `<
```

renders like this with warnings:

```
NO-BREAK SPACE: >“ “<  
NO-BREAK SPACE: >:literal:‘ ‘<
```

Other methods also don't work, e.g., substitutions are not recognized within formatted text.

It is, however, possible to use a *SPACE* (or *NO-BREAK SPACE*) enclosed in unicode characters *ZERO WIDTH SPACE*:

```
| ZERO WIDTH SPACE + SPACE + ZERO WIDTH SPACE: >` ` `<  
| ZERO WIDTH SPACE + NO-BREAK SPACE + ZERO WIDTH SPACE: >` ` `<
```

which renders like this:

ZERO WIDTH SPACE + SPACE + ZERO WIDTH SPACE: > <

ZERO WIDTH SPACE + NO-BREAK SPACE + ZERO WIDTH SPACE: > <

See also how to document a single space character within a string in reST/Sphinx?:

14.1.2 Invisible substitutions (space/blank)

Glyph	LaTeX?	Code	Unicode Name
><	✓	U+0020	SPACE
><	✓	U+00A0	NO-BREAK SPACE
><	-	U+200B	ZERO WIDTH SPACE
><	✓	U+FEFF	ZERO WIDTH NO-BREAK SPACE

These definitions can be used to add invisible breaks in the document structure:

```
.. |SPC|    unicode:: U+0020 .. SPACE
.. |NBSP|   unicode:: U+00A0 .. NO-BREAK SPACE
.. |RSPC|   replace:: :rem:`x` :rem:`x`

.. |ZSPC|   unicode:: U+200B .. ZERO WIDTH SPACE
.. |ZNBSP|  unicode:: U+FEFF .. ZERO WIDTH NO-BREAK SPACE
.. |RZSPC|  replace:: \ :rem:`x`

| > |SPC| <
| > |NBSP| <
| > |RSPC| <

| > |ZSPC| <
| > |ZNBSP| <
| > |RZSPC| <
```

Which renders as:

><
> <
><

> <
> <
><

These definitions can also be used with backslash whitespace suppression:

```
| >\ |SPC|\ <
| >\ |NBSP|\ <
| >\ |RSPC|\ <

| >\ |ZSPC|\ <
| >\ |ZNBSP|\ <
| >\ |RZSPC|\ <
```

Which renders as:

><
><

><

><

><

><

The same effect can be achieved with the `:trim:` option of the `unicode` directive:

```
| >\ |GSPC|\ <
| >\ |GZSPC|\ <

.. |GSPC| unicode:: U+0020 .. GREEDY SPACE
   :trim:
.. |GZSPC| unicode:: U+200B .. GREEDY ZERO WIDTH SPACE
   :trim:
```

Which renders as:

><

><

14.1.3 Representing space characters

Besides textual descriptions like `SPC` and `SPACE`, there are some Unicode characters:

Glyph	LaTeX?	Code	Unicode Name
☐	✓	U+2423	OPEN BOX
	✓	U+2422	BLANK SYMBOL
␣	-	U+2420	SYMBOL FOR SPACE
◻	-	U+237D	SHOULDERED OPEN BOX

See question [What character can I use to represent the space bar?](#), info about [Unicode spaces](#), [LaTeX: Explicit space character?](#)

14.1.4 LaTeX Unicode declarations

Here are LaTeX preamble declarations for the above missing unicode characters:

```
\usepackage{pmbboxdraw}
\ifdefined\DeclareUnicodeCharacter
%% ZERO WIDTH SPACE
\DeclareUnicodeCharacter{200B}{}
%% SYMBOL FOR SPACE
\DeclareUnicodeCharacter{2420}{\tiny\raisebox{1ex}[.5\ht\strutbox][0pt]{S}{P}}
%% SHOULDERED OPEN BOX
\DeclareUnicodeCharacter{237D}{\textvisiblespace}
\fi
```

These declarations are already included in the standard `sphinx-doc` framework.

14.2 Slides

Slides extensions come as HTML themes, standalone targets and for docutils only.

14.2.1 HTML theme

- `sphinxjp.themes.revealjs` - `reveal.js` presentation style theme for Sphinx

Source on [github](#) forked from `tell-k`. Using `return42` fork, since on 2019-06-21 the pull request from 2019-05-01 spanning 2 years is still open.

- Set up separate document directory, if applicable, e.g. `doc-slides`.
- The main document may be an index README or a normal chapter, e.g. `README-slides.txt`.
- In `doc-slides/Makefile` set `MAIN_DOC` to `README-slides.txt`, and `PRESENTATION` to 1:

```
PRESENTATION = 1
MAIN_DOC = README-slides.txt
```

- In `doc-slides/conf.py` activate the `revealjs` HTML theme.

Note: As of 2019-06-22, `sphinxjp.themes.revealjs` is integrated into `inst.sh` and snippets. It only needs to be activated in `Makefile` and `conf.py`.

```
pip install https://github.com/return42/sphinxjp.themes.revealjs/archive/master.zip
#pip install sphinxjp.themes.revealjs
```

in `conf.py`

```
extensions = ['sphinxjp.themes.revealjs']
html_theme = 'revealjs'
html_use_index = False
html_theme_options = {
    'width': 1920,
    'height': 1080,
    "margin": 0,
}
```

render with

```
make html
```

- `sphinxjp.themes.impressjs` provides `impressjs` directive for `impress.js` presentation control, provides `impressjs` presentation theme for rendering HTML documents.

Source on [github](#)

```
pip install sphinxjp.themes.impressjs
```

in `conf.py`

```
extensions = ['sphinxjp.themes.impressjs']
html_theme = 'sphinxjp.themes.impressjs'
html_use_index = False
```

render with

```
make html
```

- `htmlslide`

Source on [bitbucket](#)

```
pip install sphinxjp.themes.htmlslide
```

in `conf.py`

```
extensions = ['sphinxjp.themecore']
html_theme = 'htmlslide'
pygments_style = 'monokai'
```

Edit source, each section is a slide.

Render with

```
make html
```

14.2.2 Standalone target

- [Hieroglyph](#)

[Source on github](#)

```
pip install hieroglyph
```

in `conf.py`

```
extensions = ['hieroglyph']
```

render with

```
make slides
```

14.2.3 Docutils only

- [Easy Slide Shows With reST & S5](#)

14.3 Sphinx Themes

```
apt-get install python-sphinx-rtd-theme
easy_install git://github.com/snide/sphinx_rtd_theme

apt-get install python-alabaster
easy_install git://github.com/bitprophet/alabaster

apt-get install python-sphinx-bootstrap-theme
easy_install git://github.com/ryan-roemer/sphinx-bootstrap-theme

apt-get install python-guzzle-sphinx-theme
easy_install git://github.com/guzzle/guzzle_sphinx_theme

apt-get install python-openstackdocstheme
apt-get install python-cloud-sptheme
```

14.4 Graphviz Dot

The built-in [Sphinx](#) extension `sphinx.ext.graphviz` allows graphviz dot(1) graphs to be specified with the `graphviz`, `graph` and `digraph` directives in the `reStructuredText` source:


```

.. graphviz::

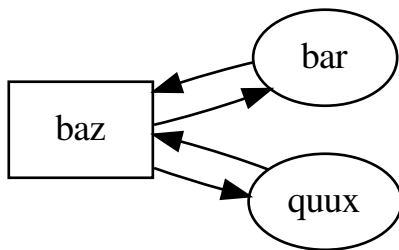
    digraph "foo" {
        rankdir=LR;

        "baz" [shape=box];

        "bar" -> "baz" -> "quux";
        "quux" -> "baz" -> "bar";
    }

```

This is rendered as:



The emacs shortcut `Ctrl-c u u d` calls the command `x-graphviz-dot-preview-current-directive`, which displays a preview of the current `graphviz` directive.

14.4.1 Graphviz dot(1) Information

Besides the [man page for dot\(1\)](#), and the [Graphviz web site](#), the [GraphViz Reference](#) is a good resource for

- [Node, Edge and Graph Attributes](#)
- supported HTML elements in [HTML-Like Labels](#) of *nodes* with shape *plaintext*.

`snippets(1)` also provides for a basic graph template with a choice of interesting options and examples:

```
snn -m dot
```

14.5 Sphinx Mercurial

See `sphinxcontrib-mercurial`.

Usage:

```

.. hg_version::

.. hg_changelog::
    :repo_root_path: ..
    :max_commits: 5
    :branch: default
    :path: some/file_or_dir

```

The updated package `python-sphinxcontrib-mercurial` is in the local package repository and can be installed with:

```
apt-get install python-sphinxcontrib.mercurial
```

Manual installation from original sources:

```
wget -q https://pypi.python.org/packages/88/3f/
↪dccabe4be8c71df0c503c0754c323a81105d35c4ba6625788eb03a2ffc04/sphinxcontrib-mercurial-0.2.
↪tar.gz
tar -zxf sphinxcontrib-mercurial-0.2.tar.gz
(
cd sphinxcontrib-mercurial-0.2/sphinxcontrib/mercurial || exit 1
patch -p0 <<'EOF'
--- hg_changelog.orig.py      2012-10-27 11:14:35.000000000 +0200
+++ hg_changelog.py 2018-04-09 11:55:41.665255313 +0200
@@ -14,12 +14,14 @@
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.

+import os
+import sys
+
+from docutils import nodes
+from docutils.parsers.rst import directives
+from sphinx.util.compat import Directive

-from mercurial import ui, hg, util
-
+from mercurial import ui, hg, util, cmdutil

class HgChangeLog(Directive):
    has_content = True
@@ -37,6 +39,11 @@
    repo_root_path = "."
    if 'repo_root_path' in self.options:
        repo_root_path = self.options['repo_root_path']
+
+    else:
+        repo_root_path = cmdutil.findrepo(os.path.abspath('.'))
+        if not repo_root_path:
+            sys.stderr.write('error: hg_changelog: no Repository found\n')
+            return []

    repo = hg.repository(ui.ui(), repo_root_path)
    l = nodes.bullet_list()
@@ -91,4 +98,3 @@
        if f.startswith(path):
            return True
        return False
-
EOF
)
cp -pr sphinxcontrib-mercurial-0.2/sphinxcontrib/mercurial /usr/lib/python2.7/dist-packages/
↪sphinxcontrib/
```

14.6 ReStructuredText and Sphinx bridge to Doxygen

Doxygen can produce XML output. This is used by the `Breathe Sphinx` extension to incorporate Doxygen output into Sphinx via directives:

```
.. doxygenclass:: Nutshell
   :project: nutshell
   :members:
```

DIAGRAM GENERATORS

EMACS VS. VI VS. ECLIPSE VS. ANYIDE

The pro's and con's in the religious editor war.

16.1 How to install latest stable Emacs in Ubuntu

Prerequisites:

```
apt-get install apel
```

For latest emacs stable releases on ubuntu (also recommended by [How to Install GNU Emacs 26.1 in Ubuntu](#)):

```
add-apt-repository ppa:kellyk/emacs
apt-get update && \
apt-get install emacs26 emacs26-el && \
update-alternatives --set emacs /usr/bin/emacs26
```

16.1.1 Setup for emacs-en-common

Integration with *emacs-en-common* is not so hard. Here are the one-time actions:

```
touch /var/lib/emacs-en-common/state/flavor/installed/emacs26
mkdir -p /etc/emacs26/site-start.d
mkdir -p /usr/share/emacs26/site-lisp
test -L /usr/share/emacs/26.*/site-lisp || \
mv /usr/share/emacs/26.*/site-lisp/* /usr/share/emacs26/site-lisp/
```

Followed by the upgrade actions, after a new emacs release was installed:

```
test -L /usr/share/emacs/26.*/site-lisp || \
rm -rf /usr/share/emacs/26.*/site-lisp
rm -f /usr/share/emacs/26.*/site-lisp
ln -s ../../emacs26/site-lisp /usr/share/emacs/26.*/
```

If `/usr/share/emacs26/site-lisp/subdirs.el` has previously been updated, it is not necessary to apply the following modifications.

Since *subdirs.el* is loaded before *site-init* is inhibited, it can be used as replacement for *startup.el*. Create `/usr/share/emacs26/site-lisp/subdirs.el`:

```
;; |:here:|
cat <<'EOF' >>/usr/share/emacs26/site-lisp/subdirs.el
(defvar package--builtin-versions
  ;; Mostly populated by loaddefs.el via autoloading-builtin-package-versions.
  (purecopy `((emacs . , (version-to-list emacs-version))))
  "Alist giving the version of each versioned builtin package.
I.e. each element of the list is of the form (NAME . VERSION) where
NAME is the package name as a symbol, and VERSION is its version")
```

(continues on next page)

Before installing the new emacs, repair package *apel*. If necessary (Ubuntu 16.04):

```
grep '[^\][\N]' /usr/share/emacs/site-lisp/apel/poe.el
```

replace `\W` in doc strings of `/usr/share/emacs/site-lisp/apel/poe.el` with `\W`:

```
vi /usr/share/emacs/site-lisp/apel/poe.el
:
1, $s, \\N, \\N, g
```

Hint: the substitution command works also fine in `sed(1)`.

16.1.4 Activate packages

Add *emacs26* to packages depending on emacs flavor. If *emacs25* is also installed from *kelleyk/emacs* repository, additionally remove *emacs25*:

```
/bin/grep --color -nH -e 'emacs2[45]' /usr/lib/emacsen-common/packages/*/* /usr/sbin/update-
↪ auctex-elisp

/usr/lib/emacsen-common/packages/install/auctex:116: (emacs24|emacs26|emacs-snapshot)
/usr/lib/emacsen-common/packages/install/psgml:50: emacs26 | emacs24)
/usr/lib/emacsen-common/packages/remove/auctex:60: (emacs24|emacs26|emacs-snapshot)
/usr/lib/emacsen-common/packages/remove/psgml:52: emacs21 | emacs22 | emacs23 | emacs24 |
↪ emacs26 | emacs-snapshot)
/usr/sbin/update-auctex-elisp:56:FLAVORS=${*:-'emacs24 emacs26 emacs-snapshot'}
```

!todo: do it manually a couple of times, then write a script (`python(1)` or `sed(1)`)

16.1.5 Optional packages

If *dvc* is installed, put this at the start of `/etc/emacs/site-start.d/50dvc.el` to activate the *ewoc* system package.

```
;; |:here:| `
(condition-case err
  (let ((load-path (reverse load-path)))
    ;; search system libraries first
    (require 'ewoc))
  (error (message "error: %s" (error-message-string err))))
;; |:here:| `
```

16.1.6 Integrate emacs into emacsen-common

Now update site-lisp with the installed debian packages:

```
/usr/lib/emacsen-common/emacs-install emacs26
```

Note: if *emacs26* was already installed before the *emacsen-common* integration was conducted, reconfigure *emacsen-common* packages:

```
dpkg-reconfigure $( ls -l /usr/lib/emacsen-common/packages/install/* | sed 's,./,,' | sort |
↪ uniq )
```

16.2 Point, mark, region, kill ring

In Emacs, the *point* is the current position of cursor:

Some text POINTat some position
continued on next line. Lorem ipsum dolor.

The key sequence C-SPC sets the position of the *mark* to the position of *point*:

Some text MARKPOINTat some position
continued on next line. Lorem ipsum dolor.

When *point* is moved, the *mark* stays in place:

Some text MARKat some position
continuedPOINT on next line. Lorem ipsum dolor.

The *region* is the text between *mark* and *point*:

at some position
continued

C-x C-x exchanges *point* and *mark*. It also activates the region:

Some text POINTat some position
continuedMARK on next line. Lorem ipsum dolor.

The *region* can always be moved or copied onto the kill ring, which acts like a clipboard with a history of killed/copied strings. If the *region* is active (highlighted), pressing BACKSPC removes the *region* without copying it to the kill ring.

Emacs	Notepad	Description
C-SPC	SHIFT	mark beginning of region
C-w	C-x	move region to kill ring
M-w	C-c	copy region to kill ring
C-y	C-v	insert last string from kill ring
M-y	•	directly after C-y or M-y replaces inserted string with previous string from kill ring
C-u - M-y	•	directly after C-y or M-y replaces inserted string with next string from kill ring

Emacs allows to handle line ends without explicitly setting the mark:

Emacs	Notepad	Description
C-k	S-end C-x	move line without line break to kill ring
C-k C-k	S-end right C-x	move line to kill ring
C-k C-k C-y	S-end right C-c	copy line to kill ring

Emulating the special *kill-line* functionality with the *region* is more involved:

Emacs	Notepad	Description
C-SPC C-e right C-w	S-end C-x	move line without line break to kill ring
C-SPC C-e right C-w	S-end right C-x	move line to kill ring
C-SPC C-e right M-w	S-end right C-c	copy line to kill ring
C-SPC C-e right BACKSPC	S-end right <ANYTHING>	delete line without affecting kill ring

Other commands for killing text entities:

Shortcut	Command	Description
M-d	M-x kill-word	kill word
C-M-k	M-x kill-sexp	kill S-expression
	M-x kill-sentence	kill sentence
	M-x kill-paragraph	kill paragraph
	M-x kill-comment	kill comment

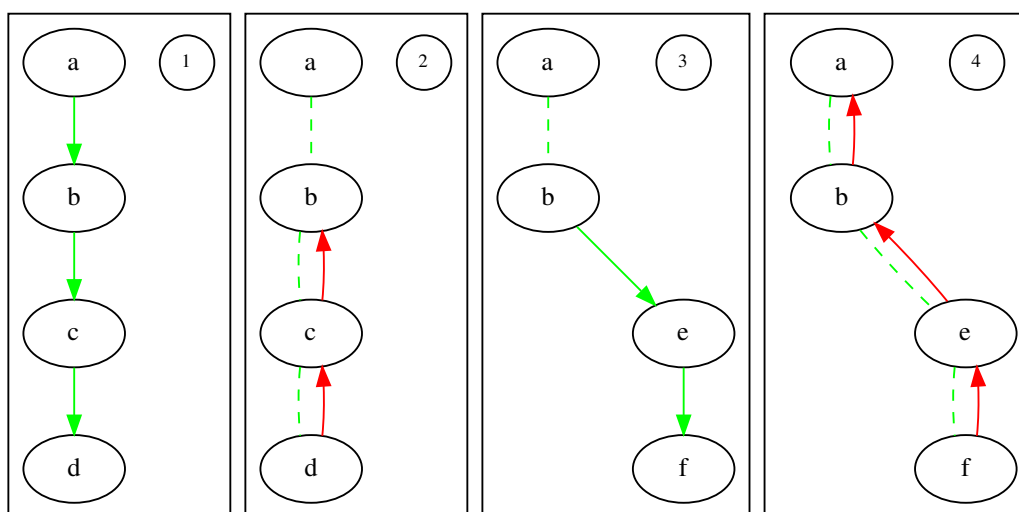
16.3 Undo

- The Emacs undo semantics is lossless.
- Vi has a lossless (and persistent) undo tree and so does Emacs (Emacs: Undo Tree package)

16.3.1 Standard Undo Function

The standard undo semantics of adding edits, undo some and – with the next edit – loose everything that was undone is just silly.

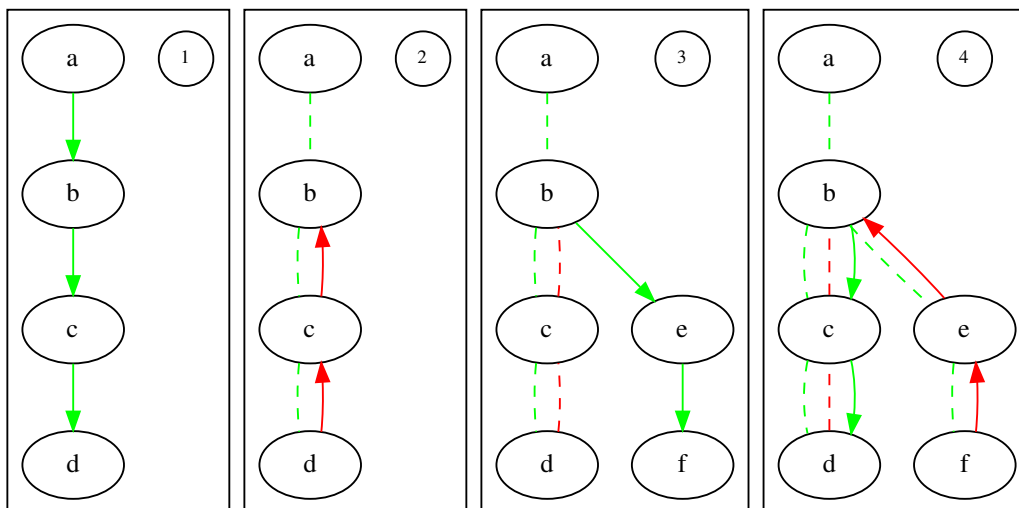
Step	Action	Description
1)	Edit	4 chunks added a, b, c, d
2)	Undo	2 chunks removed d, c
3)	Edit	2 chunks added e, f
4)	Undo	2 chunks removed f, e
	Undo	1 chunk removed b



16.3.2 Emacs Undo Function

For Emacs an undo that removes a previous edit is just another action that can be undone later.

Step	Action	Description
1)	Edit	4 chunks added a, b, c, d
2)	Undo	2 chunks removed d, c
3)	Edit	2 chunks added e, f
4)	Undo	2 chunks removed f, e
	Undo	2 chunks added c, d



16.3.3 Emacs Undo Tree

The Emacs: `Undo Tree` package realizes the previously described tree structure removing the recursion from branches. It offers a visual interface (`C-x u`) for easy navigation and fancy display of diffs .

<code>C-/</code>	<code>undo-tree-undo</code>
<code>C-__</code>	<code>undo-tree-undo</code>
<code>C-?</code>	<code>undo-tree-redo</code>
<code>C-x u</code>	<code>undo-tree-visualize</code>

16.4 Abbreviations

`vi(1)` supports abbreviations and mappings with `:abbrev` (`:iabbrev`) and `:map`.

In order to define an abbreviation local to major mode with replacement of `@here@` tag in Emacs, type:

C-SPC (if SPC @here@ SPC run-a-command) C-x a x l xxif RETURN

Both, automatic expansion in abbrev mode:

xxif SPC

and explicit expansion with inactive abbrev mode:

xxif C-x a e

expand to:

(if POINT run-a-command)

Short-cut	Command	Description
C-x a x m	M-x abbrevx-toggle-abbrev-mode	private command to toggle abbrev mode globally
C-x a x g	M-x abbrevx-add-global-abbrev	private command to define global abbreviation from current region, replacing @here@ tag on expansion
C-x a x l	M-x abbrevx-add-mode-abbrev	private command to define mode specific abbreviation from current region, replacing @here@ tag on expansion
C-x a e	M-x expand-abbrev	expand abbreviation before point
C-x a g	M-x add-global-abbrev	define global abbreviation, which works in all major modes
C-x a l	M-x add-mode-abbrev	define mode specific abbreviation, which only works in current major mode

16.5 Dynamic Abbreviation Expansion

vim(1) supports dynamic abbreviations in insert mode with C-P and C-N. vi(1) does not support this extension.

M-/ = Alt+Shift+7

Start typing a word, e.g. *dy*, then press M-/ repeatedly. **All** open buffers (not just project files) are searched for a matching expansion:

```
dynamic
dylan
dyl
dylan-console
dylan-repl
dylanlid
dylan-lid
dy
```

This is extremely useful for using long identifiers in programs:

```
VERY_LONG_IDENTIFIERS_WHICH_NOBODY_WANTS_TO_TYPE = "value"
...
VE RY_LONG_IDENTIFIERS_WHICH_NOBODY_WANTS_TO_TYPE
```

Eclipse only has a very limited functionality for mainly strongly typed, object oriented languages:

object. <popup>

16.6 Key Sequences

Vi supports key sequences as mappings (*:map*, *:imap*).

It seems possible to define key sequences in Eclipse: [Eclipse - Bindings](#).

16.7 Extensions

Vi has a macro language.

Eclipse extensions are realized via Java plugins. The extension points can then be assigned to key sequences. In order to [create a plugin](#), an entire Eclipse Java package must be created and all kinds of non-evident manual actions have to be executed. EASE makes it better with Rhino (Javascript) and Jython integration. Python is supposedly fully integrated since 2017 (not verified, not investigated). See also [section 8.7.1, Integrated Development Environment](#).

The equivalent of a “*package*” in Emacs is just a simple file (conventionally with the extension `.el` (but not necessarily)).

Let’s put it in `~/ .emacs.d` as `~/ .emacs.d/my-file.el`. The file contains an interactive function, suitable for assignment to a key sequence:

```
(defun my-func (arg)
  "Help".
  (interactive "sText: ")
  (insert (concat "/" arg "/")))
)
```

To “*install*” this “*package*”, add the following code to your `.emacs` file:

```
(load-file "~/ .emacs.d/my-file.el")
```

To assign the function `my-func ()` to a key sequence, add the following to your `.emacs` file:

```
(global-set-key "C-c uet" 'my-func)
```

You can also put this in `my-file.el` for auto-installation of the key sequence shortcut `:)`.

That’s it folks!

16.8 Tips and Tricks

:todo: English

Leerzeilen entfernen: `M-% C-q C-j C-q C-j RET C-q C-j RET`

Rechteckblock mit String auffüllen:

1. Anfang des Rechtecks markieren (`C-SPACE`)
2. Cursor am Ende des Rechtecks positionieren
3. `C-x r t <STRING> RET`

16.9 Useful packages

16.9.1 Table editing

Emacs comes with `table.el`, which allows editing ASCII tables in a WYSIWYG manner.

To recognize tables in a document, enter `M-x table-recognize RET`. Besides the *Table* menu there are shortcuts **accessible inside table cells** with the prefix `C-c C-c`. `C-c C-c C-h` **inside a table cell** show the key bindings.

!todo!

CSV

1. ProjeQtOr (Semikolon, Westeuropäisch ISO-8895-15)
 1. Öffnen in LibreOffice
 2. Speichern als XLSX
 3. Wandlung in RST-Tabelle:

```
xlsx-dump.sh --rest @datei@.xlsx
```

2. tagIDEasy (Komma, UTF-8)

kann direkt verarbeitet werden

```
xlsx-dump.sh --rest @datei@.csv
```

ID	Name	Typ	Kunde	Pro- jek- t- code	Farbe	Ende- da- tum bestätigt	Plan- Ende- datum	Fortsch- RSP datum	Stu- tus	Pro- jek- t- sta- tus	erledi- g	gesch- lossen	Hyperlink	
1	PROJECT- LEAVE- PE- RIOD	min- Es- tra- tive						0	1	recorded	0	0	http://scherer.wiedenmann.intern/projeqtor/view/main.php?directAccess=true&objectClass=Project&objectId=1	
2	Traverse	Fixed Price	Kunde1	123456		2020- 01- 23	2020- 01- 24	82	2	done	se- cured	0	0	http://scherer.wiedenmann.intern/projeqtor/view/main.php?directAccess=true&objectClass=Project&objectId=2
3	Traverse- hal- terung	Fixed Price	Kunde1			2019- 11- 16	2020- 01- 09	100	2.2	done	se- cured	1	0	http://scherer.wiedenmann.intern/projeqtor/view/main.php?directAccess=true&objectClass=Project&objectId=3
4	Meues Pro- jekt	Fixed Price					2020- 01- 16	0	3	recorded		0	0	http://scherer.wiedenmann.intern/projeqtor/view/main.php?directAccess=true&objectClass=Project&objectId=4
5	Pro- jekt_1	Fixed Price						0	4	recorded		0	0	http://scherer.wiedenmann.intern/projeqtor/view/main.php?directAccess=true&objectClass=Project&objectId=5
16.9.	Useful packages													123
6	Pro- jekt_2	Fixed Price						0	5	recorded		0	0	http://scherer.wiedenmann.intern/projeqtor/

Manuell

```
+----
| spalte 1 | xyz
| | uuu
| . | uux
```

1. Tabellenbereich markieren
2. F5 27: 27 => REGION: Run `fmt_tables -border rest on region -headlines 1`

Ergebnis:

```
+-----+
| spalte 1          | xyz          |
+-----+-----+
| cc                | uuu alskfjkl asdfkjaslkfj aklsdfj |
+-----+-----+
| aldkk dk kkkk kk | uux          |
+-----+-----+
```

Kann auch mit `M-x table-recognize` im table mode bearbeitet werden. Im table mode gilt innerhalb der Tabelle die Belegung mit Präfix `C-c C-c` (Belegungshilfe mit `C-c C-c C-h`). Table mode beenden mit `M-x table-unrecognize`.

!todo: vi: tables ??

!todo: eclipse: tables ??

16.10 Symbol tags

vi supports colon-delimited tags: `:tag:` (I just don't know how)

For emacs a generalized tagging, tag navigation and tag search package is available (*symbol-tag*).

The standard navigation is performed with `M-right` (previous tag) and `M-left` (next tag) (see [listing 16.1](#)).

listing 16.1: Basic symbol tag navigation

```

      M-up
      (here tag)
M-left          M-right
(any tag)      (any tag)
      M-down
      (here tag)
```

The here tag is a tag which is directly reachable by navigation keys `M-up` (previous tag) and `M-down` (next tag). Initially, the here tag symbol is undefined. The key sequence `M-h` sets the here tag symbol to *here*. The here tag symbol can be set to any symbol with `C-u M-up` or `C-u M-down`.

`M-i` inserts a symbol tag, `M-k` deletes the symbol tag near point.

The alternate delimiter set from a delimiter stack can be activated with `C-c C-x`. The delimiter stack and quick help is shown with `C-c C-d C-v`.

An independent delimiter is defined for enclosing symbols at point with `M-e` (`M-x symbol-tag-enclose`). The enclosing delimiter can be defined with a prefix arg selecting an entry from the delimiter set menu e.g. `C-u 1 3 M-e` (see F8).

There are key sequences available for marking the block of lines between two symbol tags with the same symbol (see [table 16.1](#)). With a prefix arg, `teh` tag symbol can be set explicitly.

table 16.1: Mark line block between two symbol tags

Shortcut	Command	w/first line	w/last line
C-c r r	M-x symbol-tag-region	•	•
C-c r 1	M-x symbol-tag-region-1	yes	•
C-c r 2	M-x symbol-tag-region-2	•	yes
C-c r 3	M-x symbol-tag-region-3	yes	yes

A `grep-find()`, which constructs a correctly quoted pattern for symbol tags is available as M-g (M-x `symbol-tag-grep-find`).

An `occur()`, which constructs a correctly quoted pattern for symbol tags is available as M-o (M-x `symbol-tag-occur`).

C-h m shows mode help. In help buffer move to section about *symbol-tag* minor-mode.

C-h k M-<down> shows help for key bound to `next-symbol-tag()`

C-h b shows current key bindings. M-x `occur RET -symbol-tag RET` in buffer **Help** shows keys for symbol movement.

16.11 Directory/filename shortcuts

shortcut	description
C-c u d d	copy directory of buffer to kill-ring
C-c u d f	basename of file, e.g. README-emacs-vi-eclipse.txt
C-u C-c u d f	with prefix: full path of file, e.g. /home/da/project/documentation/ README-emacs-vi-eclipse.txt

shortcut	command
C-c u d d	M-x <code>dired-xx-copy-directory-as-kill</code>
C-c u d o	M-x <code>dired-xx-copy-directory-as-kill-other-window</code>
C-c u d w	M-x <code>dired-xx-copy-directory-as-kill-windos</code>
C-c u d f	M-x <code>dired-xx-copy-filename-as-kill</code>

16.11.1 URL Link Generation

C-c u l r or M-x `wsx-make-link-rst` produces a reStructuredText link from a URL.

E.g., M-x `wsx-make-link-rst RET https://de.wikipedia.org/wiki/Zonentransfer RET RET` produces:

```
`Zonentransfer - Wikipedia`_
.. _`Zonentransfer - Wikipedia`: https://de.wikipedia.org/wiki/Zonentransfer
```

Reference the link as:

```
`Zonentransfer - Wikipedia`_
```

or with alternate link text:

```
`Zonentransfer (alternate link text) <Zonentransfer - Wikipedia>`_
```

E.g.:

```
Link with alternate text  
`Zonentransfer (alternate link text) <Zonentransfer - Wikipedia>`_  
referencing plain link  
`Zonentransfer - Wikipedia`_.
```

produces:

Link with alternate text [Zonentransfer \(alternate link text\)](#) referencing plain link [Zonentransfer – Wikipedia](#).

16.11.2 URL key bindings

Enter C-c u l C-h to view all bindings for URLs:

Key sequence	Elisp function
C-c u l c	wsx-make-link-cite
C-c u l d	urlx-decode-url
C-c u l e	urlx-encode-string
C-c u l h	wsx-make-link-html
C-c u l l	wsx-make-link-latex
C-c u l m	wsx-make-link-markdown-ref
C-c u l M	wsx-make-link-markdown
C-c u l p	wsx-make-link-perl
C-c u l r	wsx-make-link-rst
C-c u l R	wsx-make-link-rst-embedded
C-c u l s	wsx-make-link-snip
C-c u l w	wsx-make-link-wiki

16.11.3 Embedded link definition from separate link definition

The separated link definition is also useful to construct the embedded link variant (which is directly available as M-x wsx-make-link-rst-embedded).

1. Starting with the link definition:

```
.. _`Zonentransfer - Wikipedia`: https://de.wikipedia.org/wiki/Zonentransfer
```

2. remove dot, dot, space, underscore . . _ at start:

```
`Zonentransfer - Wikipedia`: https://de.wikipedia.org/wiki/Zonentransfer
```

3. remove backtick, colon ` : and add less-than < before URL :

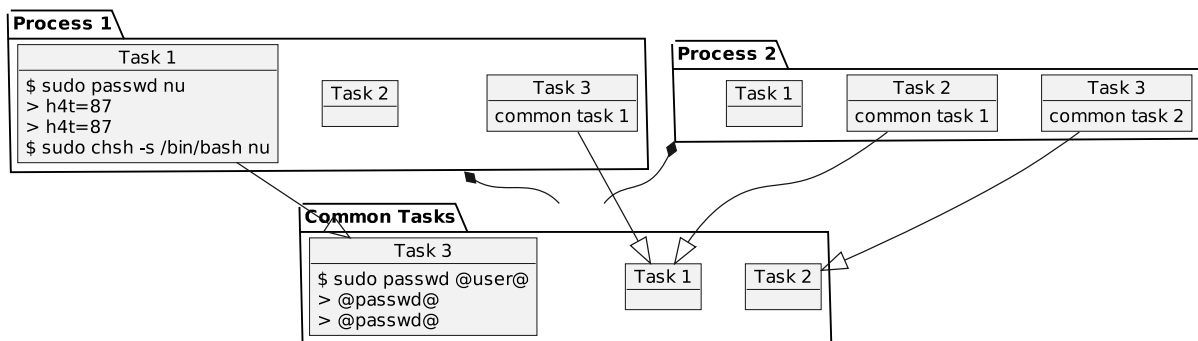
```
`Zonentransfer - Wikipedia <https://de.wikipedia.org/wiki/Zonentransfer
```

4. add greater-than, backtick, underscore >`_ at end of URL:

```
`Zonentransfer - Wikipedia <https://de.wikipedia.org/wiki/Zonentransfer>`_
```


DOCUMENT SNIPPETS

Document snippets solve the problem of describing multiple processes which consist of partial common subsets of tasks and additional unique tasks for each process. Document snippets are defined by enclosing sections in `|<-snip->|` tags. The document snippets are then inserted at places marked with a `|<-snap->|` tag. The `sphinx_doc_snip.py` tool also provides replacement facilities for creating specific versions of generic documents.



17.1 Document Snippet Definition

A document snippet is enclosed in `|<-snip->|` tags, which must start in column 0:

```
>|<-snip->|
>DOCUMENT SNIPPET BODY
>|<-snip->|
```

The tags can also be commented as `>> \||<-snip->|:`

```
>> \||<-snip->|
>>DOCUMENT SNIPPET BODY
>> \||<-snip->|
```

17.1.1 Item Prefix

An item prefix is a non-empty string at the beginning of a line, consisting of

- optional leading whitespace
- a dash `-`, plus `+`, asterisks `*`, hash dot `#.`, integer dot `1.`, or space `_`
- a minimum of 1 space `_`.

An item prefix matches the regular expression `([_*([+*_#.]|(?:[#]|[0-9]+)[.])_+)`.

When a `|<-snip->|` tag is preceded by an item prefix the end marker is recognized at the same indentation:

```
> - |<-snip->| item text ...
>   item text ...
>   .. \|<-snip->| snippet title
```

17.1.2 Item Prefix Snippet Examples

```
>- |<-snip->| some item
>
> item body filled with stuff.
> item body filled with stuff.
>
> .. \|<-snip->| item 1
>
> * |<-snip->| some indented item
>
>   indented item body filled with stuff.
>   indented item body filled with stuff.
>
> .. \|<-snip->| item 2
```

- some item
 - item body filled with stuff. item body filled with stuff.
 - some indented item
 - indented item body filled with stuff. indented item body filled with stuff.

17.1.3 Document Snippet Names

The possible names of the document snippet are derived from

1. the title given after the starting or ending `.. \|<-snip->|` marker:

```
.. \|<-snip->| Most Preferred Snippet Title
```

If the starting `|<-snip->|` tag has an item prefix, the title is only recognized for the end marker.

2. the first section title:

```
.. \|<-snip->|
-----
:rem:`||:sec:||` Snippet Section Title
-----
.. \|<-snip->|
```

3. the first label in the snippet body:

```
.. \|<-snip->|
.. _`Snippet Section Title Ref`:
.. \|<-snip->|
```

The names are used in that order as title candidates for *document snippet references*.

17.1.4 Document Snippet Flags

The first argument of the starting `.. \|<-snip->|` marker is evaluated as a string of flags consisting of:

a minus sign – The snippet is removed from document body, when parsing.

17.1.5 Document Snippet Examples

Here is a document snippet definition named `Snippet Section Title Ref`, with a section title of `Snippet Section Title` and an extra title `Snippet Section Extra Name`:

```
.. \<-snip->| - Snippet Section Extra Name
.. _`Snippet Section Title Ref`:

-----
:rem:``|:sec:|``\ Snippet Section Title
-----

section body ...

.. \<-snip->|
```

An unlabeled document snippet with section header:

```
.. \<-snip->|

-----
:rem:``|:sec:|``\ Unlabeled Snippet Title
-----

section body ...

.. \<-snip->|
```

A named document snippet without label or section header:

```
.. \<-snip->| named snippet
.. \<-snip->|
```

17.2 Snippet Tag Substitutions

The document snippet tags `<-snip->` and `<-snap->` are defined as substitutions, which are effectively blank:

```
.. <-snip->| replace:: \ :rem:`x`
.. <-snap->| replace:: \ :rem:`x`
```

17.3 Document Snippet References

A document snippet reference is recognized, when a line starts with a `<-snap->` tag:

```
><-snap->| :ref:`Snippet Tag Substitutions Label`
```

In this case, the entire snippet definition including the section header (and section label if appropriate) are substituted for the reference line.

17.3.1 Item Prefix Reference

If a `<-snap->` tag is preceded by an *item prefix*, e.g.:

```
>- <-snap->| `Snippet Tag Substitutions`_
>+ <-snap->| `Snippet Tag Substitutions`_
>* <-snap->| :ref:`Snippet Tag Substitutions Label`
>_ <-snap->| `Snippet Tag Substitutions`_
>1. <-snap->| :ref:`Snippet Tag Substitutions Label`
>#. <-snap->| :ref:`Snippet Tag Substitutions Label`
```

the snippet is inserted properly indented as item without label and section header:

```
- snippet body
  snippet body
```

A document snippet reference is also recognized as comment:

```
>.. \|<-snap->| named snippet
>.. - |<-snap->| named snippet
>.. * |<-snap->| named snippet
```

17.3.2 Item Prefix Reference Examples

Here are some live examples from the item snippets defined above. They are actually expanded in the resolved version of this document.

1. item 2
2. item 1
 - item 1
 - item 2

17.3.3 Reference Features

The tags `|<-sn0p->|`, `|<-sn1p->|`, ... `|<-sn9p->|` are equivalent to the `|<-snap->|` tag with the following additional features:

Code	Feature
0	remove item prefix, process as section
1	Bold Title
2	Bold Title, replace item bullet
3	Definition List (title = term)
4	Definition List (title = term), replace item bullet
5	Field List (title = field)
6	Field List (title = field), replace item bullet
7	undefiniert
8	undefiniert
9	undefiniert

For a snippet reference with `|<-sn1p->|` tag:

```
>* |<-sn1p->| :ref:`Snippet Tag Substitutions Label`
```

the snippet title is added as bold header:

```
* **Snippet Tag Defintions**
  snippet body
  snippet body
```

With a `|<-sn2p->|` tag:

```
>* |<-sn2p->| :ref:`Snippet Tag Substitutions Label`
```

the bullet indentation is removed and the snippet title is added as bold header:

```
**Snippet Tag Defintions**
```

```
snippet body
snippet body
```

17.3.4 Reference Feature Examples

Here are some live examples from the item snippets defined above. They are actually expanded in the resolved version of this document.

- *fixed item*

```
item 2 expanded with bold title
```

```
item 2
```

```
expanded with bold title, item removed
```

Some definitions:

```
item 2
```

```
item 1
```

Some fields:

```
item 2
```

```
item 1
```

Itemized list expanding into various things:

- item 2

```
expanding as definition, extended with some stuff
```

- item 2

```
expanding as field, extended with some stuff
```

17.3.5 Reference Headers

The tags `|<-sn#p->|`, `|<-sn=p->|`, `|<-sn-p->|`, `|<-sn~p->|`, `|<-sn.p->|`, `|<-sn+p->|`, are equivalent to the `|<-snap->|` tag, however they explicitly change the section header over-/underline. If the referenced snippet does not have a section header, it is generated from the title.

17.3.6 Reference Examples

In the resolved document, the following section is moved from above to here:

Snippet Tag Substitutions

The following item, tagged with `|<-snap->|` is either rendered as a ReST reference in the source document or the actual document snippet is inserted *without* a section title during the document snippet resolution pass:

- *Snippet Tag Substitutions*

The following item, tagged with `|<-sn1p->|` is either rendered as a ReST reference in the source document or the actual document snippet is inserted *with* a section title during the document snippet resolution pass:

Snippet Tag Substitutions

Here comes a demoted version of the section ...

Snippet Tag Substitutions

17.4 Replacement Facilities

See section *Replacements* in *sphinx_doc_snip.py* for a details.

17.4.1 Replacement Test

In the resolved version, both sides will read *laptop-mb*:

```
@hostname@ => laptop-mb
```

17.5 sphinx_doc_snip.py

sphinx_doc_snip.py - process sphinx-doc document snippets.

usage:	sphinx_doc_snip.py [OPTIONS] file-to-process ...
or	import sphinx_doc_snip

17.5.1 Options

-force	overwrite existing files.
-no-defaults	do not read default configuration files.
-c, -config CFG	add CFG to list of configuration files.
-i, -include INC	add INC as document to parse for snippets only. The document is not resolved.
-r, -replace REPL	add replacement <i>WHAT=WITH</i> . @ <i>WHAT</i> @ is replaced by string <i>WITH</i> after resolving snippets.
-f, -format FMT	format for output documents. Default: <i>{dir}/{base}-x{ext}</i> . Placeholders: <i>path</i> , <i>dir</i> , <i>file</i> , <i>base</i> , <i>ext</i> , <i>sbase</i> (secondary base of <i>base</i>), <i>sect</i> . If format is -, standard output is used.
-html	convert to HTML with sphinx-readme.sh(1)
-pdf	convert to PDF with sphinx-readme.sh(1)
-q, -quiet	suppress warnings
-v, -verbose	verbose test output
-d, -debug[=NUM]	show debug information
-h, -help	display this help message
-template list	show available templates.
-eide[=COMMMac	show IDE template list (implies -template list).
-template[=NAME]	extract named template to standard output. Default NAME is -.
-extract[=DIR]	extract adhoc files to directory DIR (default: .)
-explode[=DIR]	explode script with adhoc in directory DIR (default __adhoc__)
-setup[=install]	explode script into temporary directory and call <i>python setup.py install</i>
-implode	implode script with adhoc
-t, -test	run doc tests

17.5.2 Description

`sphinx_doc_snip.py` parses a set of documents for document snippet definitions and resolves document snippet references accordingly. The specification is in [section 17, Document Snippets](#).

Configuration Files

The configuration files

- doc/doc-snip.rc.default
- .doc-snip.rc.default
- doc/doc-snip.rc
- .doc-snip.rc

are read in that order.

Example:

```
FORCE=no
FORMAT=' {sbase} {sxt} '
INCLUDE='
README.txt
README-stations.txt
'
REPLACE='
user=Max Benutzer
user_short=mb
'
```

Replacements

The replacement facility allows to specify arbitrary replacement text for placeholders enclosed in @. With the following replacement options:

replacement option	placeholder	replacement
-replace 'user=Max Benutzer'	@user@	Max Benutzer
-replace 'user_short=mb'	@user_short@	mb
-replace 'hostname_pfx=laptop-'	@hostname_pfx@	laptop-
-replace 'hostname=@hostname_pfx@@user_short@'	@hostname@	@host- name_pfx@@user_short@

the placeholder @hostname@ is replaced recursively accordingly:

@hostname@ => *laptop-mb*

- **!todo:** recursive snippet resolution

17.5.3 Module

17.5.4 Automatic Exports

```
>>> for ex in __all__: printf(sformat('from {0} import {1}', __name__, ex))
from sphinx_doc_snip import Documentation
from sphinx_doc_snip import Document
from sphinx_doc_snip import Snippet
from sphinx_doc_snip import Section
```

17.5.5 Explicit Exports

```
>>> if '__all_internal__' in globals():
...     for ex in __all_internal__:
...         printf(sformat('from {0} import {1}', __name__, ex))
```

17.5.6 Details

```
class sphinx_doc_snip.Documentation(*args,**kwargs)
```

```
>>> _docs = Documentation()
```

```
>>> _docs.register(Document((1, 'doc_snippet_test.rst', DOC_SNIPPET_TEST)))
True
```



```

>>> _docs.dump_snippets(file=sys.stdout)
..
.. -----
.. \<-snip->| Labeled Section | Labeled Section Label
.. -----
:rem:`||:sec:||`\ Labeled Section
.. -----
Section body filled with labeled stuff.
Section body filled with labeled stuff.
Section body filled with labeled stuff.
<BLANKLINE>
field: value
field: value
..
.. -----
.. \<-snip->| - Unlabeled Section
.. -----
:rem:`||:sec:||`\ Unlabeled Section
.. -----
Section body filled with unlabeled stuff.
Section body filled with unlabeled stuff.
Section body filled with unlabeled stuff.
..
.. -----
.. \<-snip->| named paragraph
Section body filled with named stuff.
Section body filled with named stuff.
Section body filled with named stuff.
..
.. -----
.. \<-snip->| item 1
some item
<BLANKLINE>
item body filled with stuff.
item body filled with stuff.
..
.. -----
.. \<-snip->| item 2
some indented item
<BLANKLINE>
indented item body filled with stuff.
indented item body filled with stuff.

```

dump_snippets (*file=None*)

Returns self for chaining.

register (*doc_or_filename, include=None*)

Returns self for chaining.

resolve (*snippets=None, continued=None*)

Resolve snippets in all registered documents.

class sphinx_doc_snip.Document (**args, **kwargs*)

Initialize document, parse text or load from file handle/filename:

```

_doc = Document()
_doc.parse(TEXT)
_doc.load(file_handle)
_doc.load(filename)

```

Initialize document from filename (automatically loads file):

```

_doc = Document((1, 'filename'))

```

Initialize document from filename and text. Text is parsed and file is not loaded:

```

>>> _doc = Document((1, 'filename', DOC_SNIPPET_TEST))

```

```

>>> printf(_doc) #doctest: +ELLIPSIS
{
  "filename": "filename",
  "text": ".. \\|<-snip->|\n.. _Labeled Section Label:... |<-sn4p->| item 2\n |<-
↪sn6p->| item 2",
  "partition": {
    "start_rx": {...},
    "end_rx": {...},
    "this": [],
    "that": [],
    "sections": [
      ...
    ]
  },
  "body": {
    "start_rx": null,
    "end_rx": null,
    "this": [],
    "that": [],
    "sections": [
      {
        "start": ".. \\|<-snip->|",
        "body": [
          ".. _Labeled Section Label:",
          "",
          "-----",
          ":rem:`||:sec:|`\\ Labeled Section",
          "-----",
          "",
          "Section body filled with labeled stuff.",
          "Section body filled with labeled stuff.",
          "Section body filled with labeled stuff.",
          "",
          "field: value",
          "field: value"
        ],
        "end": ".. \\|<-snip->|",
        "fsep": ": ",
        "is_snippet": true,
        "title": null,
        "label": null,
        "header": null,
        "drop": null,
        "titles": [
          "Labeled Section",
          "Labeled Section Label"
        ],
        "pfx": "",
        "label_done": null
      },
      {
        "start": null,
        "body": [
          "",
          "field: value",
          "field: value",
          ""
        ],
        "end": null,
        "fsep": ": ",
        "is_snippet": null
      },
      {
        "start": null,
        "body": [
          ""
        ],
        "end": null,
        "fsep": ": ",
        "is_snippet": null
      }
    ]
  }
}

```

(continues on next page)

(continued from previous page)

```

"start": ".. \\|<-snip->| named paragraph",
"body": [
  "",
  "Section body filled with named stuff.",
  "Section body filled with named stuff.",
  "Section body filled with named stuff.",
  ""
],
"end": ".. \\|<-snip->|",
"fsep": ": ",
"is_snippet": true,
"title": null,
"label": null,
"header": null,
"drop": null,
"titles": [
  "named paragraph"
],
"pfx": "",
"label_done": null
},
{
  "start": null,
  "body": [
    ""
  ],
  "end": null,
  "fsep": ": ",
  "is_snippet": null
},
{
  "start": null,
  "body": [
    "- |<-snip->| some item",
    "",
    " item body filled with stuff.",
    " item body filled with stuff."
  ],
  "end": " .. \\|<-snip->| item 1",
  "fsep": ": ",
  "is_snippet": "-",
  "title": null,
  "label": null,
  "header": null,
  "drop": null,
  "titles": [
    "item 1"
  ],
  "pfx": "- ",
  "label_done": null
},
{
  "start": null,
  "body": [
    ""
  ],
  "end": null,
  "fsep": ": ",
  "is_snippet": null
},
{
  "start": null,
  "body": [
    " * |<-snip->| some indented item",
    "",
    " indented item body filled with stuff.",
    " indented item body filled with stuff."
  ],
  "end": " .. \\|<-snip->| item 2",
  "fsep": ": ",
  "is_snippet": "*",

```

(continues on next page)

```

        "title": null,
        "label": null,
        "header": null,
        "drop": null,
        "titles": [
            "item 2"
        ],
        "pfx": " * ",
        "label_done": null
    },
    {
        "start": null,
        "body": [
            "",
            ".. |<-snap->| named paragraph",
            "",
            "- |<-snap->| see :ref:`Labeled Section Label`",
            "- |<-snlp->| see `Unlabeled Section`_",
            "",
            "#. |<-snap->| item 2",
            "#. |<-snap->| item 1",
            "",
            "|<-snlp->| item 2",
            "",
            "|<-sn3p->| item 2",
            "|<-sn3p->| item 1",
            "",
            "|<-sn5p->| item 2",
            "|<-sn5p->| item 1",
            "",
            "|<-sn4p->| item 2",
            "|<-sn6p->| item 2"
        ],
        "end": null,
        "fsep": ": ",
        "is_snippet": null
    }
]
},
"snippets": {
    "start_rx": null,
    "end_rx": null,
    "this": [],
    "that": [],
    "sections": [
        {
            "start": null,
            "body": [
                "Section body filled with labeled stuff.",
                "Section body filled with labeled stuff.",
                "Section body filled with labeled stuff.",
                "",
                "field: value",
                "field: value"
            ],
            "end": null,
            "fsep": ": ",
            "is_snippet": true,
            "title": "Labeled Section",
            "label": null,
            "header": [
                "-----",
                ":rem:`||:sec:||`\\ Labeled Section",
                "-----"
            ],
            "drop": null,
            "titles": [
                "Labeled Section",
                "Labeled Section Label"
            ],
            "pfx": " ",

```

(continues on next page)

(continued from previous page)

```

    "label_done": null
  },
  {
    "start": null,
    "body": [
      "Section body filled with unlabeled stuff.",
      "Section body filled with unlabeled stuff.",
      "Section body filled with unlabeled stuff."
    ],
    "end": null,
    "fsep": ": ",
    "is_snippet": true,
    "title": "Unlabeled Section",
    "label": null,
    "header": [
      "-----",
      ":rem:`||:sec:|`\\ Unlabeled Section",
      "-----"
    ],
    "drop": true,
    "titles": [
      "Unlabeled Section"
    ],
    "pfx": "",
    "label_done": null
  },
  {
    "start": null,
    "body": [
      "Section body filled with named stuff.",
      "Section body filled with named stuff.",
      "Section body filled with named stuff."
    ],
    "end": null,
    "fsep": ": ",
    "is_snippet": true,
    "title": "named paragraph",
    "label": null,
    "header": null,
    "drop": null,
    "titles": [
      "named paragraph"
    ],
    "pfx": "",
    "label_done": null
  },
  {
    "start": null,
    "body": [
      "some item",
      "",
      "item body filled with stuff.",
      "item body filled with stuff."
    ],
    "end": null,
    "fsep": ": ",
    "is_snippet": "-",
    "title": "item 1",
    "label": null,
    "header": null,
    "drop": null,
    "titles": [
      "item 1"
    ],
    "pfx": "- ",
    "label_done": null
  },
  {
    "start": null,
    "body": [
      "some indented item",

```

(continues on next page)

(continued from previous page)

```

        "",
        "indented item body filled with stuff.",
        "indented item body filled with stuff."
    ],
    "end": null,
    "fsep": ": ",
    "is_snippet": "*",
    "title": "item 2",
    "label": null,
    "header": null,
    "drop": null,
    "titles": [
        "item 2"
    ],
    "pfx": " * ",
    "label_done": null
    }
}
}
"resolved": null
}

```

```

>>> _doc.dump_snippets(file=sys.stdout)
..
.. -----
.. \<-snip->| Labeled Section | Labeled Section Label
-----
:rem:`||:sec:||` Labeled Section
-----
Section body filled with labeled stuff.
Section body filled with labeled stuff.
Section body filled with labeled stuff.
<BLANKLINE>
field: value
field: value
..
.. -----
.. \<-snip->| - Unlabeled Section
-----
:rem:`||:sec:||` Unlabeled Section
-----
Section body filled with unlabeled stuff.
Section body filled with unlabeled stuff.
Section body filled with unlabeled stuff.
..
.. -----
.. \<-snip->| named paragraph
Section body filled with named stuff.
Section body filled with named stuff.
Section body filled with named stuff.
..
.. -----
.. \<-snip->| item 1
some item
<BLANKLINE>
item body filled with stuff.
item body filled with stuff.
..
.. -----
.. \<-snip->| item 2
some indented item
<BLANKLINE>
indented item body filled with stuff.
indented item body filled with stuff.

```

```

>>> printf(str(_doc.partition.sections) == DOC_SNIPPET_TEST)
True

```

```
>>> printf(str(_doc.body.sections) == DOC_SNIPPET_TEST)
False
```

```
>>> _resolved = _doc.resolve()
>>> printf(str(_resolved.sections))
.. \|-snip->|
.. _Labeled Section Label:
<BLANKLINE>
-----
:rem:`||:sec:||` Labeled Section
-----
<BLANKLINE>
Section body filled with labeled stuff.
Section body filled with labeled stuff.
Section body filled with labeled stuff.
<BLANKLINE>
field: value
field: value
.. \|-snip->|
<BLANKLINE>
field: value
field: value
<BLANKLINE>
<BLANKLINE>
.. \|-snip->| named paragraph
<BLANKLINE>
Section body filled with named stuff.
Section body filled with named stuff.
Section body filled with named stuff.
<BLANKLINE>
.. \|-snip->|
<BLANKLINE>
- \|-snip->| some item
<BLANKLINE>
    item body filled with stuff.
    item body filled with stuff.
    .. \|-snip->| item 1
<BLANKLINE>
    * \|-snip->| some indented item
<BLANKLINE>
    indented item body filled with stuff.
    indented item body filled with stuff.
    .. \|-snip->| item 2
<BLANKLINE>
<BLANKLINE>
.. Source: .. \|-snap->| named paragraph
<BLANKLINE>
Section body filled with named stuff.
Section body filled with named stuff.
Section body filled with named stuff.
<BLANKLINE>
<BLANKLINE>
- Section body filled with labeled stuff.
  Section body filled with labeled stuff.
  Section body filled with labeled stuff.
<BLANKLINE>
  field: value
  field: value
<BLANKLINE>
  .. Source: - \|-snap->| see :ref:`Labeled Section Label`
<BLANKLINE>
- **Unlabeled Section**
<BLANKLINE>
  Section body filled with unlabeled stuff.
  Section body filled with unlabeled stuff.
  Section body filled with unlabeled stuff.
<BLANKLINE>
  .. Source: - \|-snip->| see `Unlabeled Section`_
<BLANKLINE>
<BLANKLINE>
  #. some indented item
```

(continues on next page)

(continued from previous page)

```

<BLANKLINE>
    indented item body filled with stuff.
    indented item body filled with stuff.
<BLANKLINE>
    .. Source:  #. |<-snap->| item 2
<BLANKLINE>
    #. some item
<BLANKLINE>
    item body filled with stuff.
    item body filled with stuff.
<BLANKLINE>
    .. Source:  #. |<-snap->| item 1
<BLANKLINE>
<BLANKLINE>
    **item 2**
<BLANKLINE>
    some indented item
<BLANKLINE>
    indented item body filled with stuff.
    indented item body filled with stuff.
<BLANKLINE>
    .. Source:  |<-snlp->| item 2
<BLANKLINE>
<BLANKLINE>
    item 2
    some indented item
<BLANKLINE>
    indented item body filled with stuff.
    indented item body filled with stuff.
<BLANKLINE>
    .. Source:  |<-sn3p->| item 2
<BLANKLINE>
    item 1
    some item
<BLANKLINE>
    item body filled with stuff.
    item body filled with stuff.
<BLANKLINE>
    .. Source:  |<-sn3p->| item 1
<BLANKLINE>
<BLANKLINE>
    :item 2:
    some indented item
<BLANKLINE>
    indented item body filled with stuff.
    indented item body filled with stuff.
<BLANKLINE>
    .. Source:  |<-sn5p->| item 2
<BLANKLINE>
    :item 1:
    some item
<BLANKLINE>
    item body filled with stuff.
    item body filled with stuff.
<BLANKLINE>
    .. Source:  |<-sn5p->| item 1
<BLANKLINE>
<BLANKLINE>
    item 2
    some indented item
<BLANKLINE>
    indented item body filled with stuff.
    indented item body filled with stuff.
<BLANKLINE>
    .. Source:  |<-sn4p->| item 2
<BLANKLINE>
    :item 2:
    some indented item
<BLANKLINE>
    indented item body filled with stuff.
    indented item body filled with stuff.

```

(continues on next page)

(continued from previous page)

```
<BLANKLINE>
.. Source: |<-sn6p->| item 2
<BLANKLINE>
```

dump_snippets (*file=None*)

Dump document snippets.

load (*file_or_name*)

Load document from file.

Parameters *file_or_name* – file handle or filename.

map_snippets (*include=None*)

Get document snippets dictionary.

parse (*text=None, partition=None*)

Parse document for document snippets.

register_snippets_with (*snippets, include=None*)

Register document snippets in snippets dict.

Parameters *snippets* – snippets dictionary.

resolve (*snippets=None*)

Resolve snippets in document.

Returns `pyjsmo.sections.SectPart` instance with resolved body.

Parameters *snippets* – snippet section dictionary. I `None`, the document snippets are

class `sphinx_doc_snip.Snippet` (**args, **kwargs*)

Snippet section.

Behaves like a standard `Section` instance until method `prepare()` is called, which converts the section to a snippet.

dump (*file=None*)

Dump snippet.

prepare (*section=None*)

Prepare snippet.

reset ()

Reset snippet for new resolution.

class `sphinx_doc_snip.Section` (**args, **kwargs*)

Text section.

Behaves like a `pyjsmo.sections.Section`.

The method `resolve()` replaces snippet references with snippets.

resolve (*snippets*)

Resolve snippets in section body.

Slo-mo libraries.

18.1 Templating

18.1.1 Emacs

- [Macros](#)
Easy to create, hard to understand, hard to maintain
- [EmacsWiki: Category Templates](#)
- [EmacsWiki: Skeleton Mode](#)
 - used in cca extension
- [EmacsWiki: Tempo Mode](#)

Often used with abbrev-mode.

Skeleton mode

Abandoned, Skeletons are hard to create, hard to understand, hard to maintain.

HIGH CONTRAST COLORS

A palette with high contrast colors from Zeileis, Hornik and Murrell (2009): Escaping RGBland: Selecting Colors for Statistical Graphics // Computational Statistics & Data Analysis Volume 53, Issue 9, 1 July 2009, Pages 3259-3270.

19.1 High Contrast Palette with 24 Colors

High contrast colors					
#023fa5 HEX = #023fa5	#7d87b9 HEX = #7d87b9	#bec1d4 HEX = #bec1d4	#d6bcc0 HEX = #d6bcc0	#bb7784 HEX = #bb7784	#8e063b HEX = #8e063b
#4a6fe3 HEX = #4a6fe3	#8595e1 HEX = #8595e1	#b5bbe3 HEX = #b5bbe3	#e6afb9 HEX = #e6afb9	#e07b91 HEX = #e07b91	#d33f6a HEX = #d33f6a
#11c638 HEX = #11c638	#8dd593 HEX = #8dd593	#c6dec7 HEX = #c6dec7	#ead3c6 HEX = #ead3c6	#f0b98d HEX = #f0b98d	#ef9708 HEX = #ef9708
#0fcfc0 HEX = #0fcfc0	#9cded6 HEX = #9cded6	#d5eae7 HEX = #d5eae7	#f3e1eb HEX = #f3e1eb	#f6c4e1 HEX = #f6c4e1	#f79cd4 HEX = #f79cd4

19.2 Thunderbird Tags

Example for Thunderbird tags in configuration file `prefs.js`.

```
user_pref("mailnews.tags.alw.color", "#11C638");
user_pref("mailnews.tags.alw.tag", "ALW");
user_pref("mailnews.tags.da.color", "#D33F6A");
user_pref("mailnews.tags.da.tag", "DA");
user_pref("mailnews.tags.js.color", "#EF9708");
user_pref("mailnews.tags.js.tag", "JS");
user_pref("mailnews.tags.ws.color", "#4A6FE3");
user_pref("mailnews.tags.ws.tag", "WS");
user_pref("mailnews.tags.x20tdb.color", "#0FCFC0");
user_pref("mailnews.tags.x20tdb.tag", "TDB");
user_pref("mailnews.tags.x50mp.color", "#8E063B");
user_pref("mailnews.tags.x50mp.tag", "MP");
user_pref("mailnews.tags.x90af.color", "#F79CD4");
user_pref("mailnews.tags.x90af.tag", "AF");
```

19.3 High Contrast Palette with alternative X11 colors

High contrast colors with X11 alternatives 1					
#023fa5 HEX = #023fa5	#7d87b9 HEX = #7d87b9	#bec1d4 HEX = #bec1d4	#d6bcc0 HEX = #d6bcc0	#bb7784 HEX = #bb7784	#8e063b HEX = #8e063b
DodgerBlue4 HEX = #104e8b	light slate gray HEX = #778899	azure3 HEX = #c1cdcd	LavenderBlush3 HEX = #cdc1c5	rosy brown HEX = #bc8f8f	DeepPink4 HEX = #8b0a50
High contrast colors with X11 alternatives 2					
#4a6fe3 HEX = #4a6fe3	#8595e1 HEX = #8595e1	#b5bbe3 HEX = #b5bbe3	#e6afb9 HEX = #e6afb9	#e07b91 HEX = #e07b91	#d33f6a HEX = #d33f6a
royal blue HEX = #4169e1	cornflower blue HEX = #6495ed	light steel blue HEX = #b0c4de	pink2 HEX = #eea9b8	pale violet red HEX = #db7093	VioletRed3 HEX = #cd3278
High contrast colors with X11 alternatives 3					
#11c638 HEX = #11c638	#8dd593 HEX = #8dd593	#c6dec7 HEX = #c6dec7	#ead3c6 HEX = #ead3c6	#f0b98d HEX = #f0b98d	#ef9708 HEX = #ef9708
lime green HEX = #32cd32	DarkSeaGreen3 HEX = #9bcd9b	gray78 HEX = #c7c7c7	MistyRose2 HEX = #eed5d2	burlywood2 HEX = #eec591	orange2 HEX = #ee9a00
High contrast colors with X11 alternatives 4					
#0fcfc0 HEX = #0fcfc0	#9cded6 HEX = #9cded6	#d5eae7 HEX = #d5eae7	#f3e1eb HEX = #f3e1eb	#f6c4e1 HEX = #f6c4e1	#f79cd4 HEX = #f79cd4
cyan3 HEX = #00ccdcd	PaleTurquoise3 HEX = #96cdcd	LightCyan2 HEX = #d1eeee	LavenderBlush2 HEX = #eee0e5	thisite2 HEX = #eed2ee	plum HEX = #dda0dd

19.4 References

References for contrast color palettes:

- resource recommendations - Where can I find a large palette / set of contrasting colors for coloring many datasets on a plot? - Graphic Design Stack Exchange
- refers in update 1 to Zeileis, Hornik and Murrell (2009): Escaping RGBland: Selecting Colors for Statistical Graphics // Computational Statistics & Data Analysis Volume 53, Issue 9, 1 July 2009, Pages 3259-3270

X11 COLORS

PlantUML follows the [web colors](#), which incorporate a subset of [X11 color names](#). The X11 colors are especially useful to visualize color ranges.

Here are some tables of X11 color names based on the X11 `rgb.txt`. If a white bar appears next to the attribute `PUML`, the color name is not supported in [PlantUML](#) (e.g. *NavyBlue*).

When in doubt, use the hexadecimal color code.

20.1 X11 Colors - Web Colors

X11 rgb.txt - Web Colors			
Crimson (web) HEX = #dc143c PUML: Crimson	Gray (web) HEX = #808080 PUML: Gray	Green (web) HEX = #008000 PUML: Green	Indigo (web) HEX = #4b0082 PUML: Indigo
Maroon (web) HEX = #800000 PUML: Maroon	MediumPurple (web) HEX = #9370db PUML: MediumPurple	Olive (web) HEX = #808000 PUML: Olive	PaleVioletRed (web) HEX = #d87093 PUML: PaleVioletRed
Purple (web) HEX = #800080 PUML: Purple	Silver (web) HEX = #c0c0c0 PUML: Silver	Teal (web) HEX = #008080 PUML: Teal	

20.2 X11 Colors

X11 rgb.txt - Colors 1			
<p>snow</p> <p>HEX = #fffffa PURL: Snow Aliases = snow1</p>	<p>GhostWhite</p> <p>HEX = #f0f8ff PURL: GhostWhite Aliases = ghost white</p>	<p>WhiteSmoke</p> <p>HEX = #f5f5f5 PURL: WhiteSmoke Aliases = white smoke gray96 grey96</p>	<p>gainsboro</p> <p>HEX = #dcdcdc PURL: Gainsboro</p>
<p>FloralWhite</p> <p>HEX = #fffaf0 PURL: FloralWhite Aliases = floral white</p>	<p>OldLace</p> <p>HEX = #fdf5e6 PURL: OldLace Aliases = old lace</p>	<p>linen</p> <p>HEX = #faf0e6 PURL: Linen</p>	<p>AntiqueWhite</p> <p>HEX = #faebd7 PURL: AntiqueWhite Aliases = antique white</p>
<p>PapayaWhip</p> <p>HEX = #ffe4d5 PURL: PapayaWhip Aliases = papaya whip</p>	<p>BlanchedAlmond</p> <p>HEX = #ffebcd PURL: BlanchedAlmond Aliases = blanded almond</p>		
<p>bisque</p> <p>HEX = #ffe4c4 PURL: Bisque Aliases = bisque1</p>	<p>PeachPuff</p> <p>HEX = #ffdeb9 PURL: PeachPuff Aliases = peach puff PeachPuff1</p>	<p>NavajoWhite</p> <p>HEX = #ffdead PURL: NavajoWhite Aliases = navajo white NavajoWhite1</p>	<p>moccasin</p> <p>HEX = #ffe4b5 PURL: Moccasin</p>
<p>cornsilk</p> <p>HEX = #fff8dc PURL: Cornsilk Aliases = cornsilk1</p>	<p>ivory</p> <p>HEX = #fffff0 PURL: Ivory Aliases = ivory1</p>	<p>LemonChiffon</p> <p>HEX = #fffacd PURL: LemonChiffon Aliases = lemon chiffon LemonChiffon1</p>	<p>seashell</p> <p>HEX = #fff5ee PURL: SeaShell Aliases = seashell1</p>
<p>honeydew</p> <p>HEX = #f0fff0 PURL: HoneyDew Aliases = honeydew1</p>	<p>MintCream</p> <p>HEX = #f5fffa PURL: MintCream Aliases = mint cream</p>	<p>azure</p> <p>HEX = #f0ffff PURL: Azure Aliases = azure1</p>	<p>AliceBlue</p> <p>HEX = #f0f8ff PURL: AliceBlue Aliases = alice blue</p>
<p>lavender</p> <p>HEX = #e6e6fa PURL: Lavender</p>	<p>LavenderBlush</p> <p>HEX = #fff0f5 PURL: LavenderBlush Aliases = lavender blush LavenderBlush1</p>	<p>MistyRose</p> <p>HEX = #ffe4e1 PURL: MistyRose Aliases = misty rose MistyRose1</p>	
<p>white</p> <p>HEX = #ffffff PURL: White Aliases = gray100 grey100</p>	<p>black</p> <p>HEX = #000000 PURL: Black Aliases = gray0 grey0</p>		
<p>DarkSlateGray</p> <p>HEX = #2f4f4f PURL: DarkSlateGray Aliases = dark slate gray dark slate grey DarkSlateGrey</p>	<p>DimGray</p> <p>HEX = #696969 PURL: DimGray Aliases = dim gray dim grey DimGrey gray41 grey41</p>	<p>SlateGray</p> <p>HEX = #708090 PURL: SlateGray Aliases = slate gray slate grey SlateGrey</p>	<p>LightSlateGray</p> <p>HEX = #778899 PURL: LightSlateGray Aliases = light slate gray light slate grey LightSlateGrey</p>
<p>DarkGray</p> <p>HEX = #a9a9a9 PURL: DarkGray Aliases = dark grey DarkGrey dark gray</p>	<p>grey</p> <p>HEX = #bebebe PURL: #bebebe Aliases = gray</p>	<p>LightGray</p> <p>HEX = #d3d3d3 PURL: LightGray Aliases = light grey LightGrey light gray</p>	

X11 rgb.txt - Colors 2

<p>MidnightBlue</p> <p>HEX = #191970 PURL: MidnightBlue Aliases = midnight blue</p>	<p>NavyBlue</p> <p>HEX = #000080 PURL: Navy Aliases = navy navy blue</p>	<p>CornflowerBlue</p> <p>HEX = #6495ed PURL: CornflowerBlue Aliases = cornflower blue</p>	
<p>DarkSlateBlue</p> <p>HEX = #483d8b PURL: DarkSlateBlue Aliases = dark slate blue</p>	<p>SlateBlue</p> <p>HEX = #6a5acd PURL: SlateBlue Aliases = slate blue</p>	<p>MediumSlateBlue</p> <p>HEX = #7b68ee PURL: MediumSlateBlue Aliases = medium slate blue</p>	<p>LightSlateBlue</p> <p>HEX = #8470ff PURL: #8470ff Aliases = light slate blue</p>
<p>DarkBlue</p> <p>HEX = #00008b PURL: DarkBlue Aliases = blue4 dark blue</p>	<p>MediumBlue</p> <p>HEX = #0000cd PURL: MediumBlue Aliases = medium blue blue3</p>	<p>RoyalBlue</p> <p>HEX = #4169e1 PURL: RoyalBlue Aliases = royal blue</p>	<p>blue</p> <p>HEX = #0000ff PURL: Blue Aliases = blue1</p>
<p>DodgerBlue</p> <p>HEX = #1e90ff PURL: DodgerBlue Aliases = dodger blue DodgerBlue1</p>	<p>DeepSkyBlue</p> <p>HEX = #00bfff PURL: DeepSkyBlue Aliases = deep sky blue DeepSkyBlue1</p>	<p>SkyBlue</p> <p>HEX = #87ceeb PURL: SkyBlue Aliases = sky blue</p>	<p>LightSkyBlue</p> <p>HEX = #87cefa PURL: LightSkyBlue Aliases = light sky blue</p>
<p>SteelBlue</p> <p>HEX = #4682b4 PURL: SteelBlue Aliases = steel blue</p>	<p>LightSteelBlue</p> <p>HEX = #b0c4de PURL: LightSteelBlue Aliases = light steel blue</p>	<p>LightBlue</p> <p>HEX = #add8e6 PURL: LightBlue Aliases = light blue</p>	<p>PowderBlue</p> <p>HEX = #b0e0e6 PURL: PowderBlue Aliases = powder blue</p>
<p>PaleTurquoise</p> <p>HEX = #afeeee PURL: PaleTurquoise Aliases = pale turquoise</p>	<p>DarkTurquoise</p> <p>HEX = #00ced1 PURL: DarkTurquoise Aliases = dark turquoise</p>	<p>MediumTurquoise</p> <p>HEX = #48d1cc PURL: MediumTurquoise Aliases = medium turquoise</p>	<p>turquoise</p> <p>HEX = #40e0d0 PURL: Turquoise</p>
<p>DarkCyan</p> <p>HEX = #008b8b PURL: DarkCyan Aliases = cyan4 dark cyan</p>	<p>cyan</p> <p>HEX = #00ffff PURL: Aqua Aliases = cyan1</p>	<p>LightCyan</p> <p>HEX = #e0ffff PURL: LightCyan Aliases = light cyan LightCyan1</p>	<p>CadetBlue</p> <p>HEX = #5f9ea0 PURL: CadetBlue Aliases = cadet blue</p>
<p>MediumAquaMarine</p> <p>HEX = #66cdaa PURL: MediumAquaMarine Aliases = medium aquamarine aquamarine3</p>	<p>aquamarine</p> <p>HEX = #7fffd4 PURL: Aquamarine Aliases = aquamarine1</p>		

X11 rgb.txt - Colors 3

<p>DarkGreen</p> <p>HEX = #006400 PURL: DarkGreen Aliases = dark green</p>	<p>DarkOliveGreen</p> <p>HEX = #556b2f PURL: DarkOliveGreen Aliases = dark olive green</p>	<p>DarkSeaGreen</p> <p>HEX = #8fbc8f PURL: DarkSeaGreen Aliases = dark sea green</p>	<p>SeaGreen</p> <p>HEX = #2e8b57 PURL: SeaGreen Aliases = sea green SeaGreen4</p>
<p>MediumSeaGreen</p> <p>HEX = #3cb371 PURL: MediumSeaGreen Aliases = medium sea green</p>	<p>LightSeaGreen</p> <p>HEX = #20b2aa PURL: LightSeaGreen Aliases = light sea green</p>	<p>LightGreen</p> <p>HEX = #90ee90 PURL: LightGreen Aliases = PaleGreen2 light green</p>	<p>PaleGreen</p> <p>HEX = #98fb98 PURL: PaleGreen Aliases = pale green</p>
<p>SpringGreen</p> <p>HEX = #00ff7f PURL: SpringGreen Aliases = spring green SpringGreen1</p>	<p>LawnGreen</p> <p>HEX = #7fcf00 PURL: LawnGreen Aliases = lawn green</p>	<p>green</p> <p>HEX = #00ff00 PURL: Lime Aliases = green1</p>	<p>chartreuse</p> <p>HEX = #7fff00 PURL: Chartreuse Aliases = chartreuse1</p>
<p>MediumSpringGreen</p> <p>HEX = #00ffa9 PURL: MediumSpringGreen Aliases = medium spring green</p>	<p>GreenYellow</p> <p>HEX = #adff2f PURL: GreenYellow Aliases = green yellow</p>	<p>LimeGreen</p> <p>HEX = #32cd32 PURL: LimeGreen Aliases = lime green</p>	<p>YellowGreen</p> <p>HEX = #9acd32 PURL: YellowGreen Aliases = yellow green OliveDrab3</p>
<p>ForestGreen</p> <p>HEX = #228b22 PURL: ForestGreen Aliases = forest green</p>	<p>OliveDrab</p> <p>HEX = #6b8e23 PURL: OliveDrab Aliases = olive drab</p>	<p>DarkKhaki</p> <p>HEX = #bdb76b PURL: DarkKhaki Aliases = dark khaki</p>	<p>khaki</p> <p>HEX = #f0e68c PURL: Khaki</p>
<p>PaleGoldenrod</p> <p>HEX = #eee8aa PURL: PaleGoldenRod Aliases = pale goldenrod</p>	<p>LightGoldenrodYellow</p> <p>HEX = #fafad2 PURL: LightGoldenRodYellow Aliases = light goldenrod yellow</p>	<p>LightYellow</p> <p>HEX = #ffffe0 PURL: LightYellow Aliases = light yellow LightYellow1</p>	<p>yellow</p> <p>HEX = #ffff00 PURL: Yellow Aliases = yellow1</p>
<p>gold</p> <p>HEX = #ffd700 PURL: Gold Aliases = gold1</p>	<p>LightGoldenrod</p> <p>HEX = #eedd82 PURL: #eedd82 Aliases = light goldenrod</p>	<p>goldenrod</p> <p>HEX = #daa520 PURL: GoldenRod</p>	<p>DarkGoldenrod</p> <p>HEX = #b8860b PURL: DarkGoldenRod Aliases = dark goldenrod</p>
<p>RosyBrown</p> <p>HEX = #bc8f8f PURL: RosyBrown Aliases = rosy brown</p>	<p>IndianRed</p> <p>HEX = #cd5c5c PURL: IndianRed Aliases = indian red</p>	<p>SaddleBrown</p> <p>HEX = #8b4513 PURL: SaddleBrown Aliases = saddle brown chocolate4</p>	<p>sienna</p> <p>HEX = #a0522d PURL: Sienna</p>
<p>peru</p> <p>HEX = #cd853f PURL: Peru Aliases = tan3</p>	<p>burlywood</p> <p>HEX = #deb887 PURL: BurlyWood</p>	<p>beige</p> <p>HEX = #f5f5dc PURL: Beige</p>	<p>wheat</p> <p>HEX = #f5deb3 PURL: Wheat</p>
<p>SandyBrown</p> <p>HEX = #f4a460 PURL: SandyBrown Aliases = sandy brown</p>	<p>tan</p> <p>HEX = #d2b48c PURL: Tan</p>	<p>chocolate</p> <p>HEX = #d2691e PURL: Chocolate</p>	<p>firebrick</p> <p>HEX = #b22222 PURL: FireBrick</p>
<p>brown</p> <p>HEX = #a52a2a PURL: Brown</p>	<p>DarkSalmon</p> <p>HEX = #e9967a PURL: DarkSalmon Aliases = dark salmon</p>	<p>salmon</p> <p>HEX = #fa8072 PURL: Salmon</p>	<p>LightSalmon</p> <p>HEX = #ffa07a PURL: LightSalmon Aliases = light salmon LightSalmon1</p>

X11 rgb.txt - Colors 4

<p>orange</p> <p>HEX = #ffa500 PURL: Orange Aliases = orange1</p>	<p>DarkOrange</p> <p>HEX = #ff8c00 PURL: Darkorange Aliases = dark orange</p>	<p>coral</p> <p>HEX = #ff7f50 PURL: Coral</p>	<p>LightCoral</p> <p>HEX = #f08080 PURL: LightCoral Aliases = light coral</p>
<p>tomato</p> <p>HEX = #ff6347 PURL: Tomato Aliases = tomato1</p>	<p>OrangeRed</p> <p>HEX = #ff4500 PURL: OrangeRed Aliases = orange red OrangeRed1</p>	<p>red</p> <p>HEX = #ff0000 PURL: Red Aliases = red1</p>	<p>DarkRed</p> <p>HEX = #8b0000 PURL: DarkRed Aliases = red4 dark red</p>
<p>HotPink</p> <p>HEX = #ff69b4 PURL: HotPink Aliases = hot pink</p>	<p>DeepPink</p> <p>HEX = #ff1493 PURL: DeepPink Aliases = deep pink DeepPink1</p>	<p>pink</p> <p>HEX = #ffc0cb PURL: Pink</p>	<p>LightPink</p> <p>HEX = #ffb6c1 PURL: LightPink Aliases = light pink</p>
<p>PaleVioletRed</p> <p>HEX = #db7093 PURL: #db7093 Aliases = pale violet red</p>	<p>maroon</p> <p>HEX = #b03060 PURL: #b03060</p>	<p>MediumVioletRed</p> <p>HEX = #c71585 PURL: MediumVioletRed Aliases = medium violet red</p>	<p>VioletRed</p> <p>HEX = #d02090 PURL: #d02090 Aliases = violet red</p>
<p>DarkMagenta</p> <p>HEX = #8b008b PURL: DarkMagenta Aliases = magenta4 dark magenta</p>	<p>magenta</p> <p>HEX = #ff00ff PURL: Fuchsia Aliases = magenta1</p>	<p>violet</p> <p>HEX = #ee82ee PURL: Violet</p>	<p>plum</p> <p>HEX = #dda0dd PURL: Plum</p>
<p>orchid</p> <p>HEX = #da70d6 PURL: Orchid</p>	<p>MediumOrchid</p> <p>HEX = #ba55d3 PURL: MediumOrchid Aliases = medium orchid</p>	<p>DarkOrchid</p> <p>HEX = #9932cc PURL: DarkOrchid Aliases = dark orchid</p>	<p>DarkViolet</p> <p>HEX = #9400d3 PURL: DarkViolet Aliases = dark violet</p>
<p>BlueViolet</p> <p>HEX = #8a2be2 PURL: BlueViolet Aliases = blue violet</p>	<p>purple</p> <p>HEX = #a020f0 PURL: #a020f0</p>	<p>MediumPurple</p> <p>HEX = #9370db PURL: #9370db Aliases = medium purple</p>	<p>thistle</p> <p>HEX = #d8bfd8 PURL: Thistle</p>

20.3 X11 Colors - Gradients

X11 rgb.txt - Gradients 1			
<p>snow1</p> <p>HEX = #fffffa PURL: Snow Aliases = snow</p>	<p>snow2</p> <p>HEX = #eee9e9 PURL: #eee9e9</p>	<p>snow3</p> <p>HEX = #cdc9c9 PURL: #cdc9c9</p>	<p>snow4</p> <p>HEX = #8b8989 PURL: #8b8989</p>
<p>seashell1</p> <p>HEX = #fff5ee PURL: SeaShell Aliases = seashell</p>	<p>seashell2</p> <p>HEX = #eee5de PURL: #eee5de</p>	<p>seashell3</p> <p>HEX = #cdc5bf PURL: #cdc5bf</p>	<p>seashell4</p> <p>HEX = #8b8682 PURL: #8b8682</p>
<p>AntiqueWhite1</p> <p>HEX = #ffefdb PURL: #ffefdb</p>	<p>AntiqueWhite2</p> <p>HEX = #eedfcc PURL: #eedfcc</p>	<p>AntiqueWhite3</p> <p>HEX = #cdb0b0 PURL: #cdb0b0</p>	<p>AntiqueWhite4</p> <p>HEX = #8b8378 PURL: #8b8378</p>
<p>bisque1</p> <p>HEX = #ffe4c4 PURL: Bisque Aliases = bisque</p>	<p>bisque2</p> <p>HEX = #eed5b7 PURL: #eed5b7</p>	<p>bisque3</p> <p>HEX = #cdb79e PURL: #cdb79e</p>	<p>bisque4</p> <p>HEX = #8b7d6b PURL: #8b7d6b</p>
<p>PeachPuff</p> <p>HEX = #ffdab9 PURL: PeachPuff Aliases = peach puff PeachPuff1</p>	<p>PeachPuff2</p> <p>HEX = #eecbad PURL: #eecbad</p>	<p>PeachPuff3</p> <p>HEX = #cdaf95 PURL: #cdaf95</p>	<p>PeachPuff4</p> <p>HEX = #8b7765 PURL: #8b7765</p>
<p>NavajoWhite</p> <p>HEX = #ffdead PURL: NavajoWhite Aliases = navajo white NavajoWhite1</p>	<p>NavajoWhite2</p> <p>HEX = #eecfa1 PURL: #eecfa1</p>	<p>NavajoWhite3</p> <p>HEX = #cdb38b PURL: #cdb38b</p>	<p>NavajoWhite4</p> <p>HEX = #8b795e PURL: #8b795e</p>
<p>LemonChiffon</p> <p>HEX = #ffffac PURL: LemonChiffon Aliases = lemon chiffon LemonChiffon1</p>	<p>LemonChiffon2</p> <p>HEX = #eee9bf PURL: #eee9bf</p>	<p>LemonChiffon3</p> <p>HEX = #cdc9a5 PURL: #cdc9a5</p>	<p>LemonChiffon4</p> <p>HEX = #8b8970 PURL: #8b8970</p>
<p>cornsilk1</p> <p>HEX = #fff8dc PURL: Cornsilk Aliases = comsilk</p>	<p>cornsilk2</p> <p>HEX = #eee8cd PURL: #eee8cd</p>	<p>cornsilk3</p> <p>HEX = #cdc8b1 PURL: #cdc8b1</p>	<p>cornsilk4</p> <p>HEX = #8b8878 PURL: #8b8878</p>
<p>ivory1</p> <p>HEX = #fffff0 PURL: Ivory Aliases = ivory</p>	<p>ivory2</p> <p>HEX = #e0eee0 PURL: #e0eee0</p>	<p>ivory3</p> <p>HEX = #c0cdc0 PURL: #c0cdc0</p>	<p>ivory4</p> <p>HEX = #8b8883 PURL: #8b8883</p>
<p>honeydew1</p> <p>HEX = #f0ffff PURL: HoneyDew Aliases = honeydew</p>	<p>honeydew2</p> <p>HEX = #e0eeee PURL: #e0eeee</p>	<p>honeydew3</p> <p>HEX = #c1cdc1 PURL: #c1cdc1</p>	<p>honeydew4</p> <p>HEX = #838b83 PURL: #838b83</p>

X11 rgb.txt - Gradients 2

<p>LavenderBlush</p> <p>HEX = #fff0f5 PURL: LavenderBlush Aliases = lavender blush LavenderBlush1</p>	<p>LavenderBlush2</p> <p>HEX = #eee0e5 PURL: #eee0e5</p>	<p>LavenderBlush3</p> <p>HEX = #cdc1c5 PURL: #cdc1c5</p>	<p>LavenderBlush4</p> <p>HEX = #8b8386 PURL: #8b8386</p>
<p>MistyRose</p> <p>HEX = #ffe4e1 PURL: MistyRose Aliases = misty rose MistyRose1</p>	<p>MistyRose2</p> <p>HEX = #eed5d2 PURL: #eed5d2</p>	<p>MistyRose3</p> <p>HEX = #cdb7b5 PURL: #cdb7b5</p>	<p>MistyRose4</p> <p>HEX = #8b7d7b PURL: #8b7d7b</p>
<p>azure1</p> <p>HEX = #f0ffff PURL: Azure Aliases = azure</p>	<p>azure2</p> <p>HEX = #e0eeee PURL: #e0eeee</p>	<p>azure3</p> <p>HEX = #c1cddc PURL: #c1cddc</p>	<p>azure4</p> <p>HEX = #838b8b PURL: #838b8b</p>
<p>SlateBlue1</p> <p>HEX = #836fff PURL: #836fff</p>	<p>SlateBlue2</p> <p>HEX = #7a67ee PURL: #7a67ee</p>	<p>SlateBlue3</p> <p>HEX = #6959cd PURL: #6959cd</p>	<p>SlateBlue4</p> <p>HEX = #473c8b PURL: #473c8b</p>
<p>RoyalBlue1</p> <p>HEX = #4876ff PURL: #4876ff</p>	<p>RoyalBlue2</p> <p>HEX = #436eee PURL: #436eee</p>	<p>RoyalBlue3</p> <p>HEX = #3a5fcd PURL: #3a5fcd</p>	<p>RoyalBlue4</p> <p>HEX = #27408b PURL: #27408b</p>
<p>blue1</p> <p>HEX = #0000ff PURL: Blue Aliases = blue</p>	<p>blue2</p> <p>HEX = #0000ee PURL: #0000ee</p>	<p>MediumBlue</p> <p>HEX = #0000cd PURL: MediumBlue Aliases = medium blue blue3</p>	<p>DarkBlue</p> <p>HEX = #00008b PURL: DarkBlue Aliases = blue4 dark blue</p>
<p>DodgerBlue</p> <p>HEX = #1e90ff PURL: DodgerBlue Aliases = dodger blue DodgerBlue1</p>	<p>DodgerBlue2</p> <p>HEX = #1c86ee PURL: #1c86ee</p>	<p>DodgerBlue3</p> <p>HEX = #1874cd PURL: #1874cd</p>	<p>DodgerBlue4</p> <p>HEX = #104e8b PURL: #104e8b</p>
<p>SteelBlue1</p> <p>HEX = #63b8ff PURL: #63b8ff</p>	<p>SteelBlue2</p> <p>HEX = #5cacee PURL: #5cacee</p>	<p>SteelBlue3</p> <p>HEX = #4f94cd PURL: #4f94cd</p>	<p>SteelBlue4</p> <p>HEX = #36648b PURL: #36648b</p>
<p>DeepSkyBlue</p> <p>HEX = #00bfff PURL: DeepSkyBlue Aliases = deep sky blue DeepSkyBlue1</p>	<p>DeepSkyBlue2</p> <p>HEX = #00b2ee PURL: #00b2ee</p>	<p>DeepSkyBlue3</p> <p>HEX = #009acd PURL: #009acd</p>	<p>DeepSkyBlue4</p> <p>HEX = #00688b PURL: #00688b</p>
<p>SkyBlue1</p> <p>HEX = #87ceff PURL: #87ceff</p>	<p>SkyBlue2</p> <p>HEX = #7ec0ee PURL: #7ec0ee</p>	<p>SkyBlue3</p> <p>HEX = #66a6cd PURL: #66a6cd</p>	<p>SkyBlue4</p> <p>HEX = #4a708b PURL: #4a708b</p>

X11 rgb.txt - Gradients 3

<p>LightSkyBlue1</p> <p>HEX = #b0e2ff PURL: #b0e2ff</p>	<p>LightSkyBlue2</p> <p>HEX = #a4d3ee PURL: #a4d3ee</p>	<p>LightSkyBlue3</p> <p>HEX = #8db6cd PURL: #8db6cd</p>	<p>LightSkyBlue4</p> <p>HEX = #607b8b PURL: #607b8b</p>
<p>SlateGray1</p> <p>HEX = #c6e2ff PURL: #c6e2ff</p>	<p>SlateGray2</p> <p>HEX = #b9d3ee PURL: #b9d3ee</p>	<p>SlateGray3</p> <p>HEX = #9fb6cd PURL: #9fb6cd</p>	<p>SlateGray4</p> <p>HEX = #6c7b8b PURL: #6c7b8b</p>
<p>LightSteelBlue1</p> <p>HEX = #cae1ff PURL: #cae1ff</p>	<p>LightSteelBlue2</p> <p>HEX = #bcd2ee PURL: #bcd2ee</p>	<p>LightSteelBlue3</p> <p>HEX = #a2b5cd PURL: #a2b5cd</p>	<p>LightSteelBlue4</p> <p>HEX = #6e7b8b PURL: #6e7b8b</p>
<p>LightBlue1</p> <p>HEX = #bfefff PURL: #bfefff</p>	<p>LightBlue2</p> <p>HEX = #b2dfee PURL: #b2dfee</p>	<p>LightBlue3</p> <p>HEX = #9ac8cd PURL: #9ac8cd</p>	<p>LightBlue4</p> <p>HEX = #68838b PURL: #68838b</p>
<p>LightCyan</p> <p>HEX = #e0ffff PURL: LightCyan Aliases = light cyan LightCyan1</p>	<p>LightCyan2</p> <p>HEX = #d1eeee PURL: #d1eeee</p>	<p>LightCyan3</p> <p>HEX = #b4cdcd PURL: #b4cdcd</p>	<p>LightCyan4</p> <p>HEX = #7a8b8b PURL: #7a8b8b</p>
<p>PaleTurquoise1</p> <p>HEX = #bbffff PURL: #bbffff</p>	<p>PaleTurquoise2</p> <p>HEX = #aeeeee PURL: #aeeeee</p>	<p>PaleTurquoise3</p> <p>HEX = #96cdcd PURL: #96cdcd</p>	<p>PaleTurquoise4</p> <p>HEX = #668b8b PURL: #668b8b</p>
<p>CadetBlue1</p> <p>HEX = #98f5ff PURL: #98f5ff</p>	<p>CadetBlue2</p> <p>HEX = #8ee5ee PURL: #8ee5ee</p>	<p>CadetBlue3</p> <p>HEX = #7ac5cd PURL: #7ac5cd</p>	<p>CadetBlue4</p> <p>HEX = #53868b PURL: #53868b</p>
<p>turquoise1</p> <p>HEX = #00f5ff PURL: #00f5ff</p>	<p>turquoise2</p> <p>HEX = #00e5ee PURL: #00e5ee</p>	<p>turquoise3</p> <p>HEX = #00c5cd PURL: #00c5cd</p>	<p>turquoise4</p> <p>HEX = #00868b PURL: #00868b</p>
<p>cyan1</p> <p>HEX = #00ffff PURL: Aqua Aliases = cyan</p>	<p>cyan2</p> <p>HEX = #00eeee PURL: #00eeee</p>	<p>cyan3</p> <p>HEX = #00cdcd PURL: #00cdcd</p>	<p>DarkCyan</p> <p>HEX = #008b8b PURL: DarkCyan Aliases = cyan4 dark cyan</p>
<p>DarkSlateGray1</p> <p>HEX = #97ffff PURL: #97ffff</p>	<p>DarkSlateGray2</p> <p>HEX = #8deeee PURL: #8deeee</p>	<p>DarkSlateGray3</p> <p>HEX = #79cdcd PURL: #79cdcd</p>	<p>DarkSlateGray4</p> <p>HEX = #528b8b PURL: #528b8b</p>

X11 rgb.txt - Gradients 4

<p>aquamarine1</p> <p>HEX = #7fffd4 PURL: Aquamarine Aliases = aquamarine</p>	<p>aquamarine2</p> <p>HEX = #76eec6 PURL: #76eec6</p>	<p>aquamarine3</p> <p>HEX = #66cd9a PURL: MediumAquaMarine Aliases = medium aquamarine MediumAquaMarine</p>	<p>aquamarine4</p> <p>HEX = #458b74 PURL: #458b74</p>
<p>DarkSeaGreen1</p> <p>HEX = #c1ffc1 PURL: #c1ffc1</p>	<p>DarkSeaGreen2</p> <p>HEX = #b4eeb4 PURL: #b4eeb4</p>	<p>DarkSeaGreen3</p> <p>HEX = #9bcd9b PURL: #9bcd9b</p>	<p>DarkSeaGreen4</p> <p>HEX = #698b69 PURL: #698b69</p>
<p>SeaGreen1</p> <p>HEX = #54ff9f PURL: #54ff9f</p>	<p>SeaGreen2</p> <p>HEX = #4eee94 PURL: #4eee94</p>	<p>SeaGreen3</p> <p>HEX = #43cd80 PURL: #43cd80</p>	<p>SeaGreen</p> <p>HEX = #2e8b57 PURL: SeaGreen Aliases = sea green SeaGreen4</p>
<p>PaleGreen1</p> <p>HEX = #9aff9a PURL: #9aff9a</p>	<p>LightGreen</p> <p>HEX = #90ee90 PURL: LightGreen Aliases = PaleGreen2 light green</p>	<p>PaleGreen3</p> <p>HEX = #7ccd7c PURL: #7ccd7c</p>	<p>PaleGreen4</p> <p>HEX = #548b54 PURL: #548b54</p>
<p>SpringGreen</p> <p>HEX = #00ff7f PURL: SpringGreen Aliases = spring green SpringGreen1</p>	<p>SpringGreen2</p> <p>HEX = #90ee76 PURL: #90ee76</p>	<p>SpringGreen3</p> <p>HEX = #00cd66 PURL: #00cd66</p>	<p>SpringGreen4</p> <p>HEX = #008b45 PURL: #008b45</p>
<p>green1</p> <p>HEX = #00ff00 PURL: Lime Aliases = green</p>	<p>green2</p> <p>HEX = #00ee00 PURL: #00ee00</p>	<p>green3</p> <p>HEX = #00cd00 PURL: #00cd00</p>	<p>green4</p> <p>HEX = #008b00 PURL: #008b00</p>
<p>chartreuse1</p> <p>HEX = #7fff00 PURL: Chartreuse Aliases = chartreuse</p>	<p>chartreuse2</p> <p>HEX = #76ee00 PURL: #76ee00</p>	<p>chartreuse3</p> <p>HEX = #66cd00 PURL: #66cd00</p>	<p>chartreuse4</p> <p>HEX = #458b00 PURL: #458b00</p>
<p>OliveDrab1</p> <p>HEX = #c0ff3e PURL: #c0ff3e</p>	<p>OliveDrab2</p> <p>HEX = #b3ee3a PURL: #b3ee3a</p>	<p>YellowGreen</p> <p>HEX = #9acd32 PURL: YellowGreen Aliases = yellow green OliveDrab3</p>	<p>OliveDrab4</p> <p>HEX = #698b22 PURL: #698b22</p>
<p>DarkOliveGreen1</p> <p>HEX = #caff70 PURL: #caff70</p>	<p>DarkOliveGreen2</p> <p>HEX = #bcee68 PURL: #bcee68</p>	<p>DarkOliveGreen3</p> <p>HEX = #a2cd5a PURL: #a2cd5a</p>	<p>DarkOliveGreen4</p> <p>HEX = #6e8b3d PURL: #6e8b3d</p>

X11 rgb.txt - Gradients 5

<p>khaki1</p> <p>HEX = #fff68f PURL: #fff68f</p>	<p>khaki2</p> <p>HEX = #eee685 PURL: #eee685</p>	<p>khaki3</p> <p>HEX = #cdc673 PURL: #cdc673</p>	<p>khaki4</p> <p>HEX = #8b864e PURL: #8b864e</p>
<p>LightGoldenrod1</p> <p>HEX = #ffec8b PURL: #ffec8b</p>	<p>LightGoldenrod2</p> <p>HEX = #eedc82 PURL: #eedc82</p>	<p>LightGoldenrod3</p> <p>HEX = #cdbe70 PURL: #cdbe70</p>	<p>LightGoldenrod4</p> <p>HEX = #8b814c PURL: #8b814c</p>
<p>LightYellow</p> <p>HEX = #ffffe0 PURL: LightYellow Aliases = light yellow LightYellow1</p>	<p>LightYellow2</p> <p>HEX = #eeeed1 PURL: #eeeed1</p>	<p>LightYellow3</p> <p>HEX = #cdcdb4 PURL: #cdcdb4</p>	<p>LightYellow4</p> <p>HEX = #8b8b7a PURL: #8b8b7a</p>
<p>yellow1</p> <p>HEX = #ffff00 PURL: Yellow Aliases = yellow</p>	<p>yellow2</p> <p>HEX = #eeee00 PURL: #eeee00</p>	<p>yellow3</p> <p>HEX = #cdcd00 PURL: #cdcd00</p>	<p>yellow4</p> <p>HEX = #8b8b00 PURL: #8b8b00</p>
<p>gold1</p> <p>HEX = #ffd700 PURL: Gold Aliases = gold</p>	<p>gold2</p> <p>HEX = #eec900 PURL: #eec900</p>	<p>gold3</p> <p>HEX = #cdad00 PURL: #cdad00</p>	<p>gold4</p> <p>HEX = #8b7500 PURL: #8b7500</p>
<p>goldenrod1</p> <p>HEX = #ffc125 PURL: #ffc125</p>	<p>goldenrod2</p> <p>HEX = #eeb422 PURL: #eeb422</p>	<p>goldenrod3</p> <p>HEX = #cd9b1d PURL: #cd9b1d</p>	<p>goldenrod4</p> <p>HEX = #8b6914 PURL: #8b6914</p>
<p>DarkGoldenrod1</p> <p>HEX = #ffb90f PURL: #ffb90f</p>	<p>DarkGoldenrod2</p> <p>HEX = #ead0e PURL: #ead0e</p>	<p>DarkGoldenrod3</p> <p>HEX = #cd950c PURL: #cd950c</p>	<p>DarkGoldenrod4</p> <p>HEX = #8b6508 PURL: #8b6508</p>
<p>RosyBrown1</p> <p>HEX = #ffc1c1 PURL: #ffc1c1</p>	<p>RosyBrown2</p> <p>HEX = #eeb4b4 PURL: #eeb4b4</p>	<p>RosyBrown3</p> <p>HEX = #cd9b9b PURL: #cd9b9b</p>	<p>RosyBrown4</p> <p>HEX = #8b6969 PURL: #8b6969</p>
<p>IndianRed1</p> <p>HEX = #ff6a6a PURL: #ff6a6a</p>	<p>IndianRed2</p> <p>HEX = #ee6363 PURL: #ee6363</p>	<p>IndianRed3</p> <p>HEX = #cd5555 PURL: #cd5555</p>	<p>IndianRed4</p> <p>HEX = #8b3a3a PURL: #8b3a3a</p>
<p>sienna1</p> <p>HEX = #ff8247 PURL: #ff8247</p>	<p>sienna2</p> <p>HEX = #ee7942 PURL: #ee7942</p>	<p>sienna3</p> <p>HEX = #cd6839 PURL: #cd6839</p>	<p>sienna4</p> <p>HEX = #8b4726 PURL: #8b4726</p>

X11 rgb.txt - Gradients 6

<p>burlywood1</p> <p>HEX = #ffd39b PURL: #ffd39b</p>	<p>burlywood2</p> <p>HEX = #eec591 PURL: #eec591</p>	<p>burlywood3</p> <p>HEX = #cdaa7d PURL: #cdaa7d</p>	<p>burlywood4</p> <p>HEX = #8b7355 PURL: #8b7355</p>
<p>wheat1</p> <p>HEX = #ffe7ba PURL: #ffe7ba</p>	<p>wheat2</p> <p>HEX = #eed8ae PURL: #eed8ae</p>	<p>wheat3</p> <p>HEX = #cdba96 PURL: #cdba96</p>	<p>wheat4</p> <p>HEX = #8b7e66 PURL: #8b7e66</p>
<p>tan1</p> <p>HEX = #ffa54f PURL: #ffa54f</p>	<p>tan2</p> <p>HEX = #ee9a49 PURL: #ee9a49</p>	<p>tan3</p> <p>HEX = #cd853f PURL: Peru Aliases = peru</p>	<p>tan4</p> <p>HEX = #8b5a2b PURL: #8b5a2b</p>
<p>chocolate1</p> <p>HEX = #ff7f24 PURL: #ff7f24</p>	<p>chocolate2</p> <p>HEX = #ee7621 PURL: #ee7621</p>	<p>chocolate3</p> <p>HEX = #cd661d PURL: #cd661d</p>	<p>SaddleBrown</p> <p>HEX = #8b4513 PURL: SaddleBrown Aliases = saddle brown chocolate4</p>
<p>firebrick1</p> <p>HEX = #ff3030 PURL: #ff3030</p>	<p>firebrick2</p> <p>HEX = #ee2c2c PURL: #ee2c2c</p>	<p>firebrick3</p> <p>HEX = #cd2626 PURL: #cd2626</p>	<p>firebrick4</p> <p>HEX = #8b1a1a PURL: #8b1a1a</p>
<p>brown1</p> <p>HEX = #ff4040 PURL: #ff4040</p>	<p>brown2</p> <p>HEX = #ee3b3b PURL: #ee3b3b</p>	<p>brown3</p> <p>HEX = #cd3333 PURL: #cd3333</p>	<p>brown4</p> <p>HEX = #8b2323 PURL: #8b2323</p>
<p>salmon1</p> <p>HEX = #ff8c69 PURL: #ff8c69</p>	<p>salmon2</p> <p>HEX = #ee8262 PURL: #ee8262</p>	<p>salmon3</p> <p>HEX = #cd7054 PURL: #cd7054</p>	<p>salmon4</p> <p>HEX = #8b4c39 PURL: #8b4c39</p>
<p>LightSalmon</p> <p>HEX = #ffa07a PURL: LightSalmon Aliases = light salmon LightSalmon1</p>	<p>LightSalmon2</p> <p>HEX = #ee9572 PURL: #ee9572</p>	<p>LightSalmon3</p> <p>HEX = #cd8162 PURL: #cd8162</p>	<p>LightSalmon4</p> <p>HEX = #8b5742 PURL: #8b5742</p>
<p>orange1</p> <p>HEX = #ffa500 PURL: Orange Aliases = orange</p>	<p>orange2</p> <p>HEX = #ee9a00 PURL: #ee9a00</p>	<p>orange3</p> <p>HEX = #cd8500 PURL: #cd8500</p>	<p>orange4</p> <p>HEX = #8b5a00 PURL: #8b5a00</p>
<p>DarkOrange1</p> <p>HEX = #ff7f00 PURL: #ff7f00</p>	<p>DarkOrange2</p> <p>HEX = #ee7600 PURL: #ee7600</p>	<p>DarkOrange3</p> <p>HEX = #cd6600 PURL: #cd6600</p>	<p>DarkOrange4</p> <p>HEX = #8b4500 PURL: #8b4500</p>

X11 rgb.txt - Gradients 7

<p>coral1</p> <p>HEX = #ff7256 PURL: #ff7256</p>	<p>coral2</p> <p>HEX = #ee6a50 PURL: #ee6a50</p>	<p>coral3</p> <p>HEX = #cd5b45 PURL: #cd5b45</p>	<p>coral4</p> <p>HEX = #8b3e2f PURL: #8b3e2f</p>
<p>tomato1</p> <p>HEX = #ff6347 PURL: Tomato Aliases = tomato</p>	<p>tomato2</p> <p>HEX = #ee5c42 PURL: #ee5c42</p>	<p>tomato3</p> <p>HEX = #cd4f39 PURL: #cd4f39</p>	<p>tomato4</p> <p>HEX = #8b3626 PURL: #8b3626</p>
<p>OrangeRed</p> <p>HEX = #ff4500 PURL: OrangeRed Aliases = orange red OrangeRed1</p>	<p>OrangeRed2</p> <p>HEX = #ee4000 PURL: #ee4000</p>	<p>OrangeRed3</p> <p>HEX = #cd3700 PURL: #cd3700</p>	<p>OrangeRed4</p> <p>HEX = #8b2500 PURL: #8b2500</p>
<p>red1</p> <p>HEX = #ff0000 PURL: Red Aliases = red</p>	<p>red2</p> <p>HEX = #ee0000 PURL: #ee0000</p>	<p>red3</p> <p>HEX = #cd0000 PURL: #cd0000</p>	<p>DarkRed</p> <p>HEX = #8b0000 PURL: DarkRed Aliases = red4 dark red</p>
<p>DebianRed</p> <p>HEX = #d70751 PURL: #d70751</p>			
<p>DeepPink</p> <p>HEX = #ff1493 PURL: DeepPink Aliases = deep pink DeepPink1</p>	<p>DeepPink2</p> <p>HEX = #ee1289 PURL: #ee1289</p>	<p>DeepPink3</p> <p>HEX = #cd1076 PURL: #cd1076</p>	<p>DeepPink4</p> <p>HEX = #8b0a50 PURL: #8b0a50</p>
<p>HotPink1</p> <p>HEX = #ff6eb4 PURL: #ff6eb4</p>	<p>HotPink2</p> <p>HEX = #ee6aa7 PURL: #ee6aa7</p>	<p>HotPink3</p> <p>HEX = #cd6090 PURL: #cd6090</p>	<p>HotPink4</p> <p>HEX = #8b3a62 PURL: #8b3a62</p>
<p>pink1</p> <p>HEX = #ffb5c5 PURL: #ffb5c5</p>	<p>pink2</p> <p>HEX = #eea9b8 PURL: #eea9b8</p>	<p>pink3</p> <p>HEX = #cd919e PURL: #cd919e</p>	<p>pink4</p> <p>HEX = #8b636c PURL: #8b636c</p>
<p>LightPink1</p> <p>HEX = #ffaeb9 PURL: #ffaeb9</p>	<p>LightPink2</p> <p>HEX = #eea2ad PURL: #eea2ad</p>	<p>LightPink3</p> <p>HEX = #cd8c95 PURL: #cd8c95</p>	<p>LightPink4</p> <p>HEX = #8b5f65 PURL: #8b5f65</p>
<p>PaleVioletRed1</p> <p>HEX = #ff82ab PURL: #ff82ab</p>	<p>PaleVioletRed2</p> <p>HEX = #ee799f PURL: #ee799f</p>	<p>PaleVioletRed3</p> <p>HEX = #cd6889 PURL: #cd6889</p>	<p>PaleVioletRed4</p> <p>HEX = #8b475d PURL: #8b475d</p>

X11 rgb.txt - Gradients 8

<p>maroon1</p> <p>HEX = #ff34b3 PURL: #ff34b3</p> 	<p>maroon2</p> <p>HEX = #ee30a7 PURL: #ee30a7</p> 	<p>maroon3</p> <p>HEX = #cd2990 PURL: #cd2990</p> 	<p>maroon4</p> <p>HEX = #8b1c62 PURL: #8b1c62</p> 
<p>VioletRed1</p> <p>HEX = #ff3e96 PURL: #ff3e96</p> 	<p>VioletRed2</p> <p>HEX = #ee3a8c PURL: #ee3a8c</p> 	<p>VioletRed3</p> <p>HEX = #cd3278 PURL: #cd3278</p> 	<p>VioletRed4</p> <p>HEX = #8b2252 PURL: #8b2252</p> 
<p>magenta1</p> <p>HEX = #ff00ff PURL: Fuchsia Aliases = magenta</p> 	<p>magenta2</p> <p>HEX = #ee00ee PURL: #ee00ee</p> 	<p>magenta3</p> <p>HEX = #cd00cd PURL: #cd00cd</p> 	<p>DarkMagenta</p> <p>HEX = #8b008b PURL: DarkMagenta Aliases = magenta4 dark magenta</p> 
<p>orchid1</p> <p>HEX = #ff83fa PURL: #ff83fa</p> 	<p>orchid2</p> <p>HEX = #ee7ae9 PURL: #ee7ae9</p> 	<p>orchid3</p> <p>HEX = #cd69c9 PURL: #cd69c9</p> 	<p>orchid4</p> <p>HEX = #8b4789 PURL: #8b4789</p> 
<p>plum1</p> <p>HEX = #ffbfff PURL: #ffbfff</p> 	<p>plum2</p> <p>HEX = #eeaeae PURL: #eeaeae</p> 	<p>plum3</p> <p>HEX = #cd96cd PURL: #cd96cd</p> 	<p>plum4</p> <p>HEX = #8b668b PURL: #8b668b</p> 
<p>MediumOrchid1</p> <p>HEX = #e066ff PURL: #e066ff</p> 	<p>MediumOrchid2</p> <p>HEX = #d15fee PURL: #d15fee</p> 	<p>MediumOrchid3</p> <p>HEX = #b452cd PURL: #b452cd</p> 	<p>MediumOrchid4</p> <p>HEX = #7a378b PURL: #7a378b</p> 
<p>DarkOrchid1</p> <p>HEX = #bf3eff PURL: #bf3eff</p> 	<p>DarkOrchid2</p> <p>HEX = #b23aee PURL: #b23aee</p> 	<p>DarkOrchid3</p> <p>HEX = #9a32cd PURL: #9a32cd</p> 	<p>DarkOrchid4</p> <p>HEX = #68228b PURL: #68228b</p> 
<p>purple1</p> <p>HEX = #9b30ff PURL: #9b30ff</p> 	<p>purple2</p> <p>HEX = #912cee PURL: #912cee</p> 	<p>purple3</p> <p>HEX = #7d26cd PURL: #7d26cd</p> 	<p>purple4</p> <p>HEX = #551a8b PURL: #551a8b</p> 
<p>MediumPurple1</p> <p>HEX = #ab82ff PURL: #ab82ff</p> 	<p>MediumPurple2</p> <p>HEX = #9f79ee PURL: #9f79ee</p> 	<p>MediumPurple3</p> <p>HEX = #8968cd PURL: #8968cd</p> 	<p>MediumPurple4</p> <p>HEX = #5d478b PURL: #5d478b</p> 
<p>thistle1</p> <p>HEX = #ffe1ff PURL: #ffe1ff</p> 	<p>thistle2</p> <p>HEX = #eed2ee PURL: #eed2ee</p> 	<p>thistle3</p> <p>HEX = #cdb5cd PURL: #cdb5cd</p> 	<p>thistle4</p> <p>HEX = #8b7b8b PURL: #8b7b8b</p> 

20.4 X11 Colors - Grey Scale

X11 rgb.txt - Grey Scale 1			
<p>grey0</p> <p>HEX = #000000 PURL: Black Aliases = black grey0</p>	<p>grey1</p> <p>HEX = #080808 PURL: #080808 Aliases = gray1</p>	<p>grey2</p> <p>HEX = #0f0f0f PURL: #0f0f0f Aliases = gray2</p>	<p>grey3</p> <p>HEX = #161616 PURL: #080808 Aliases = gray3</p>
<p>grey4</p> <p>HEX = #1d1d1d PURL: #0a0a0a Aliases = gray4</p>	<p>grey5</p> <p>HEX = #242424 PURL: #0d0d0d Aliases = gray5</p>	<p>grey6</p> <p>HEX = #2b2b2b PURL: #0f0f0f Aliases = gray6</p>	<p>grey7</p> <p>HEX = #323232 PURL: #121212 Aliases = gray7</p>
<p>grey8</p> <p>HEX = #3a3a3a PURL: #141414 Aliases = gray8</p>	<p>grey9</p> <p>HEX = #414141 PURL: #171717 Aliases = gray9</p>	<p>grey10</p> <p>HEX = #494949 PURL: #1a1a1a Aliases = gray10</p>	<p>grey11</p> <p>HEX = #505050 PURL: #1c1c1c Aliases = gray11</p>
<p>grey12</p> <p>HEX = #585858 PURL: #1f1f1f Aliases = gray12</p>	<p>grey13</p> <p>HEX = #5f5f5f PURL: #212121 Aliases = gray13</p>	<p>grey14</p> <p>HEX = #666666 PURL: #242424 Aliases = gray14</p>	<p>grey15</p> <p>HEX = #6d6d6d PURL: #262626 Aliases = gray15</p>
<p>grey16</p> <p>HEX = #747474 PURL: #292929 Aliases = gray16</p>	<p>grey17</p> <p>HEX = #7c7c7c PURL: #2b2b2b Aliases = gray17</p>	<p>grey18</p> <p>HEX = #838383 PURL: #2e2e2e Aliases = gray18</p>	<p>grey19</p> <p>HEX = #8c8c8c PURL: #303030 Aliases = gray19</p>
<p>grey20</p> <p>HEX = #929292 PURL: #333333 Aliases = gray20</p>	<p>grey21</p> <p>HEX = #9a9a9a PURL: #363636 Aliases = gray21</p>	<p>grey22</p> <p>HEX = #a1a1a1 PURL: #383838 Aliases = gray22</p>	<p>grey23</p> <p>HEX = #a9a9a9 PURL: #3b3b3b Aliases = gray23</p>
<p>grey24</p> <p>HEX = #b0b0b0 PURL: #3d3d3d Aliases = gray24</p>	<p>grey25</p> <p>HEX = #b8b8b8 PURL: #404040 Aliases = gray25</p>	<p>grey26</p> <p>HEX = #c0c0c0 PURL: #424242 Aliases = gray26</p>	<p>grey27</p> <p>HEX = #c7c7c7 PURL: #454545 Aliases = gray27</p>
<p>grey28</p> <p>HEX = #ced1d1 PURL: #474747 Aliases = gray28</p>	<p>grey29</p> <p>HEX = #d4d4d4 PURL: #4a4a4a Aliases = gray29</p>	<p>grey30</p> <p>HEX = #d9d9d9 PURL: #4d4d4d Aliases = gray30</p>	<p>grey31</p> <p>HEX = #e0e0e0 PURL: #4f4f4f Aliases = gray31</p>
<p>grey32</p> <p>HEX = #e6e6e6 PURL: #525252 Aliases = gray32</p>	<p>grey33</p> <p>HEX = #eaeaea PURL: #545454 Aliases = gray33</p>	<p>grey34</p> <p>HEX = #f0f0f0 PURL: #575757 Aliases = gray34</p>	<p>grey35</p> <p>HEX = #f5f5f5 PURL: #595959 Aliases = gray35</p>
<p>grey36</p> <p>HEX = #f9f9f9 PURL: #5c5c5c Aliases = gray36</p>	<p>grey37</p> <p>HEX = #f9f9f9 PURL: #5e5e5e Aliases = gray37</p>	<p>grey38</p> <p>HEX = #f9f9f9 PURL: #616161 Aliases = gray38</p>	<p>grey39</p> <p>HEX = #f9f9f9 PURL: #636363 Aliases = gray39</p>

X11 rgb.txt - Grey Scale 2

<p>grey40</p> <p>HEX = #666666 PURL: #666666 Aliases = gray40</p>	<p>DimGray</p> <p>HEX = #696969 PURL: DimGray Aliases = dim gray dim grey DimGrey gray41 grey41</p>	<p>grey42</p> <p>HEX = #6b6b6b PURL: #6b6b6b Aliases = gray42</p>	<p>grey43</p> <p>HEX = #6e6e6e PURL: #6e6e6e Aliases = gray43</p>
<p>grey44</p> <p>HEX = #707070 PURL: #707070 Aliases = gray44</p>	<p>grey45</p> <p>HEX = #737373 PURL: #737373 Aliases = gray45</p>	<p>grey46</p> <p>HEX = #757575 PURL: #757575 Aliases = gray46</p>	<p>grey47</p> <p>HEX = #787878 PURL: #787878 Aliases = gray47</p>
<p>grey48</p> <p>HEX = #7a7a7a PURL: #7a7a7a Aliases = gray48</p>	<p>grey49</p> <p>HEX = #7d7d7d PURL: #7d7d7d Aliases = gray49</p>	<p>grey50</p> <p>HEX = #7f7f7f PURL: #7f7f7f Aliases = gray50</p>	<p>grey51</p> <p>HEX = #828282 PURL: #828282 Aliases = gray51</p>
<p>grey52</p> <p>HEX = #858585 PURL: #858585 Aliases = gray52</p>	<p>grey53</p> <p>HEX = #878787 PURL: #878787 Aliases = gray53</p>	<p>grey54</p> <p>HEX = #8a8a8a PURL: #8a8a8a Aliases = gray54</p>	<p>grey55</p> <p>HEX = #8c8c8c PURL: #8c8c8c Aliases = gray55</p>
<p>grey56</p> <p>HEX = #8f8f8f PURL: #8f8f8f Aliases = gray56</p>	<p>grey57</p> <p>HEX = #919191 PURL: #919191 Aliases = gray57</p>	<p>grey58</p> <p>HEX = #949494 PURL: #949494 Aliases = gray58</p>	<p>grey59</p> <p>HEX = #969696 PURL: #969696 Aliases = gray59</p>
<p>grey60</p> <p>HEX = #999999 PURL: #999999 Aliases = gray60</p>	<p>grey61</p> <p>HEX = #9c9c9c PURL: #9c9c9c Aliases = gray61</p>	<p>grey62</p> <p>HEX = #9e9e9e PURL: #9e9e9e Aliases = gray62</p>	<p>grey63</p> <p>HEX = #a1a1a1 PURL: #a1a1a1 Aliases = gray63</p>
<p>grey64</p> <p>HEX = #a3a3a3 PURL: #a3a3a3 Aliases = gray64</p>	<p>grey65</p> <p>HEX = #a6a6a6 PURL: #a6a6a6 Aliases = gray65</p>	<p>grey66</p> <p>HEX = #a8a8a8 PURL: #a8a8a8 Aliases = gray66</p>	<p>grey67</p> <p>HEX = #ababab PURL: #ababab Aliases = gray67</p>
<p>grey68</p> <p>HEX = #adadad PURL: #adadad Aliases = gray68</p>	<p>grey69</p> <p>HEX = #b0b0b0 PURL: #b0b0b0 Aliases = gray69</p>	<p>grey70</p> <p>HEX = #b3b3b3 PURL: #b3b3b3 Aliases = gray70</p>	<p>grey71</p> <p>HEX = #b5b5b5 PURL: #b5b5b5 Aliases = gray71</p>
<p>grey72</p> <p>HEX = #b8b8b8 PURL: #b8b8b8 Aliases = gray72</p>	<p>grey73</p> <p>HEX = #bababa PURL: #bababa Aliases = gray73</p>	<p>grey74</p> <p>HEX = #bdbdbd PURL: #bdbdbd Aliases = gray74</p>	<p>grey75</p> <p>HEX = #bfbfbf PURL: #bfbfbf Aliases = gray75</p>
<p>grey76</p> <p>HEX = #c2c2c2 PURL: #c2c2c2 Aliases = gray76</p>	<p>grey77</p> <p>HEX = #c4c4c4 PURL: #c4c4c4 Aliases = gray77</p>	<p>grey78</p> <p>HEX = #c7c7c7 PURL: #c7c7c7 Aliases = gray78</p>	<p>grey79</p> <p>HEX = #c9c9c9 PURL: #c9c9c9 Aliases = gray79</p>
<p>grey80</p> <p>HEX = #cccccc PURL: #cccccc Aliases = gray80</p>	<p>grey81</p> <p>HEX = #cfcfcf PURL: #cfcfcf Aliases = gray81</p>	<p>grey82</p> <p>HEX = #d1d1d1 PURL: #d1d1d1 Aliases = gray82</p>	<p>grey83</p> <p>HEX = #d4d4d4 PURL: #d4d4d4 Aliases = gray83</p>

X11 rgb.txt - Grey Scale 3

<p>grey84</p> <p>HEX = #d6d6d6 PUML: #d6d6d6 Aliases = gray84</p>	<p>grey85</p> <p>HEX = #d9d9d9 PUML: #d9d9d9 Aliases = gray85</p>	<p>grey86</p> <p>HEX = #dbdbdb PUML: #dbdbdb Aliases = gray86</p>	<p>grey87</p> <p>HEX = #dedede PUML: #dedede Aliases = gray87</p>
<p>grey88</p> <p>HEX = #e0e0e0 PUML: #e0e0e0 Aliases = gray88</p>	<p>grey89</p> <p>HEX = #e3e3e3 PUML: #e3e3e3 Aliases = gray89</p>	<p>grey90</p> <p>HEX = #e5e5e5 PUML: #e5e5e5 Aliases = gray90</p>	<p>grey91</p> <p>HEX = #e8e8e8 PUML: #e8e8e8 Aliases = gray91</p>
<p>grey92</p> <p>HEX = #ebebeb PUML: #ebebeb Aliases = gray92</p>	<p>grey93</p> <p>HEX = #ededed PUML: #ededed Aliases = gray93</p>	<p>grey94</p> <p>HEX = #f0f0f0 PUML: #f0f0f0 Aliases = gray94</p>	<p>grey95</p> <p>HEX = #f2f2f2 PUML: #f2f2f2 Aliases = gray95</p>
<p>WhiteSmoke</p> <p>HEX = #f5f5f5 PUML: WhiteSmoke Aliases = white smoke gray96 grey96</p>	<p>grey97</p> <p>HEX = #f7f7f7 PUML: #f7f7f7 Aliases = gray97</p>	<p>grey98</p> <p>HEX = #fafafa PUML: #fafafa Aliases = gray98</p>	<p>grey99</p> <p>HEX = #fcfcfc PUML: #fcfcfc Aliases = gray99</p>
<p>grey100</p> <p>HEX = #ffffff PUML: White Aliases = white gray100</p>			

KNOWLEDGE ORGANIZATION

The “nice” mathematical way: a tree, a forest. Universal and more realistic (but still mathematical): a graph. Also mathematical, but useless: Venn diagrams.

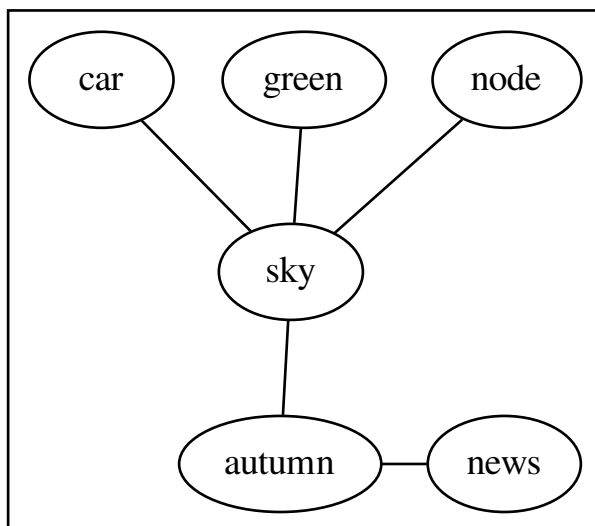
21.1 Trees

21.1.1 Definition of a Tree

The most useful definition of a [tree](#), comes from [graph theory](#), where

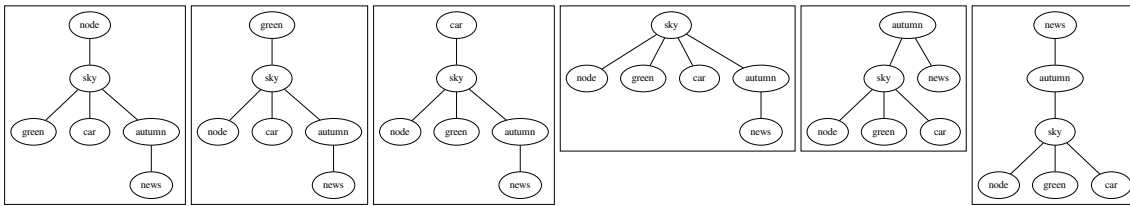
a tree is an undirected graph in which any two vertices are connected by exactly one path. Every acyclic connected graph is a tree, and vice versa. A forest is a disjoint union of trees, or equivalently an acyclic graph that is not necessarily connected.

Strangely enough, many examples on Wikipedia use numbered nodes, which implies some kind of order. That is an invalid generalization from the set of “*unfounded brain-dead assumptions*”



Mathematically, graph theory identifies all kinds of different trees with fancy names and classifications. Then there is a whole lot of more fancy properties for [trees in computer science](#). Most of these differences are members of the set of “*distinctions that the world don't need*”.

E.g., a **rooted tree** is defined as “a tree in which one vertex has been designated the root”. It is easy to see, that in a tree graph, any node can be designated as *root*. So, there is really nothing special about a *rooted tree* or a *root* node.



The distinction between undirected and directed graphs is pretty much useless and therefore in the set of “*distinctions that the world don’t need*”.

21.1.2 Walking Around a Tree

Using a *rooted tree*, there are two basic ways for **tree traversal**:

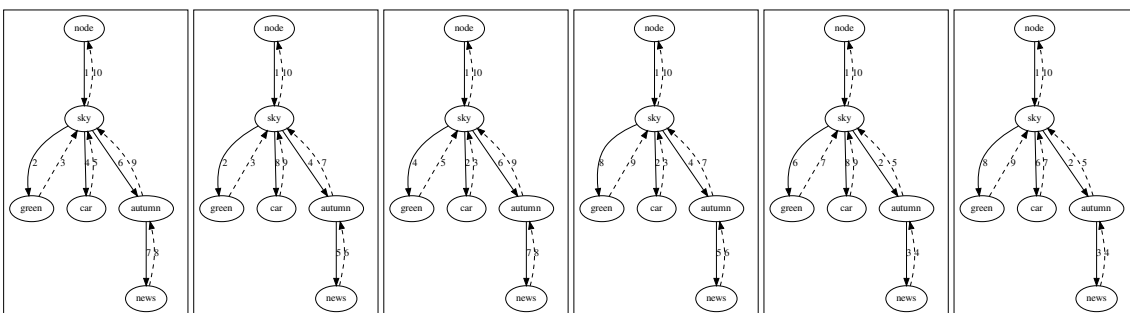
- **depth-first search** and
- **breadth-first search**.

There are many variations and combinations of these two basic tree walks, which are also in the set of “*distinctions that the world don’t need*”.

Depth-First search

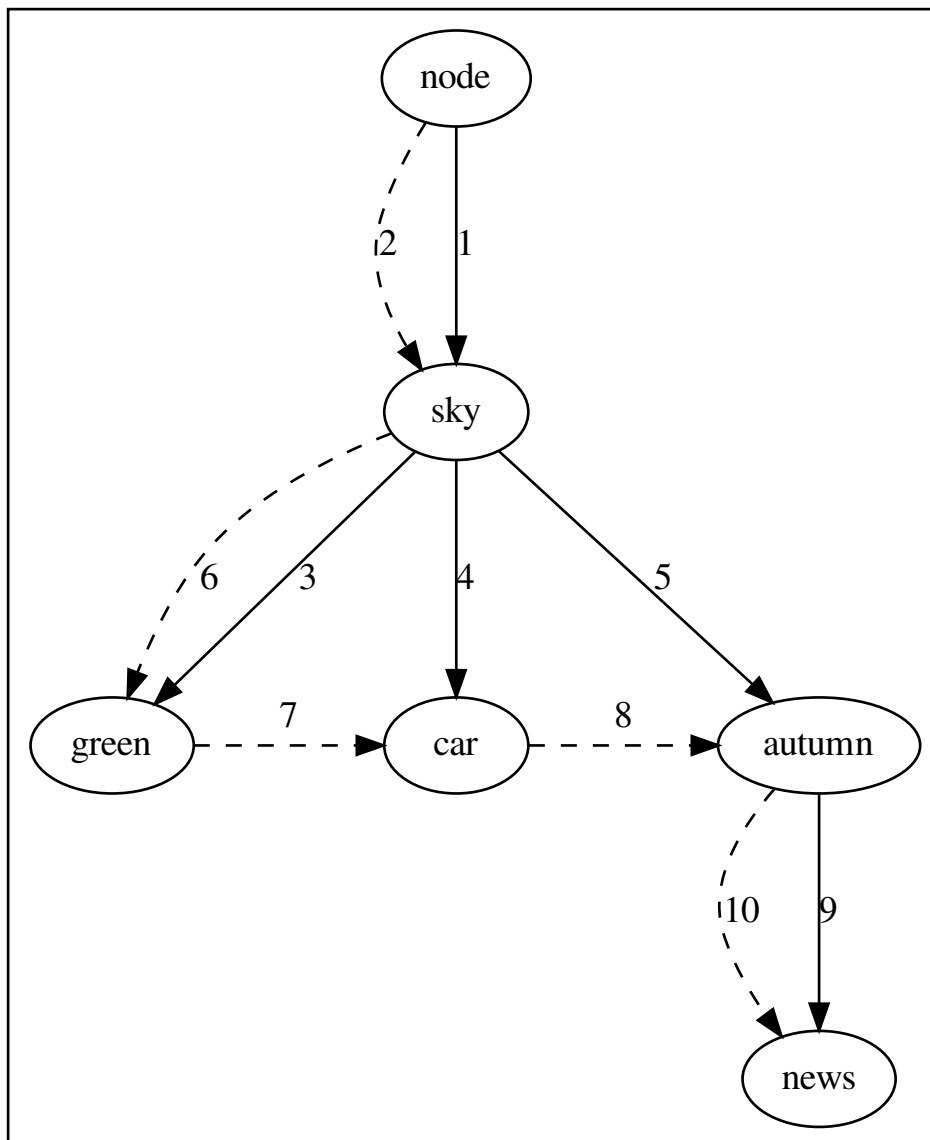
The most common way to walk around a tree is a **depth-first search**. This **tree traversal** is also presented with an implicit order of nodes, which is unnecessary and therefore belongs to the set of *unfounded brain-dead assumptions*.

Here are some valid examples of unordered **depth-first searches**, solid lines signify the first visit, dashed lines signify backtracking:



Breadth-First search

Here are is an examples of an unordered **breadth-first search**, solid lines signify the first visit, dashed lines signify backtracking:



Illustrative Video

There is a YouTube video, which illustrates [Depth First Search vs Breadth First Search](#) using a stack and a queue for backtracking. The [local copy \(subtitles\)](#) is copyright 2017 by O'NeillCode on YouTube:

Connect with me on LinkedIn! <https://www.linkedin.com/in/stephenaoneill/>

This video will go over Depth First Search vs Breadth First Search of a graph. It will explore how stacks and queues are used in the two algorithms and walk through it.

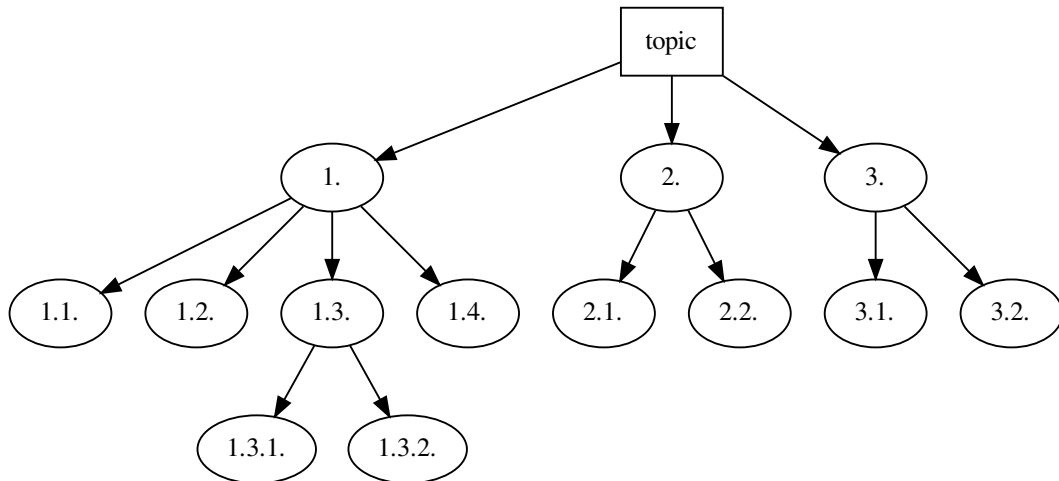
If you like the video, give it a thumbs up! If you want to see more videos from me hit the subscribe button! All code can be found here: <https://github.com/oneillcode/YouTube-Tutorials>

O'Neill Code's purpose is to provide in-depth tutorials on a wide range of programming material. It

can focus on interview questions or simple algorithms depending on the user requests. If you have any video requests please let me!

21.1.3 Books

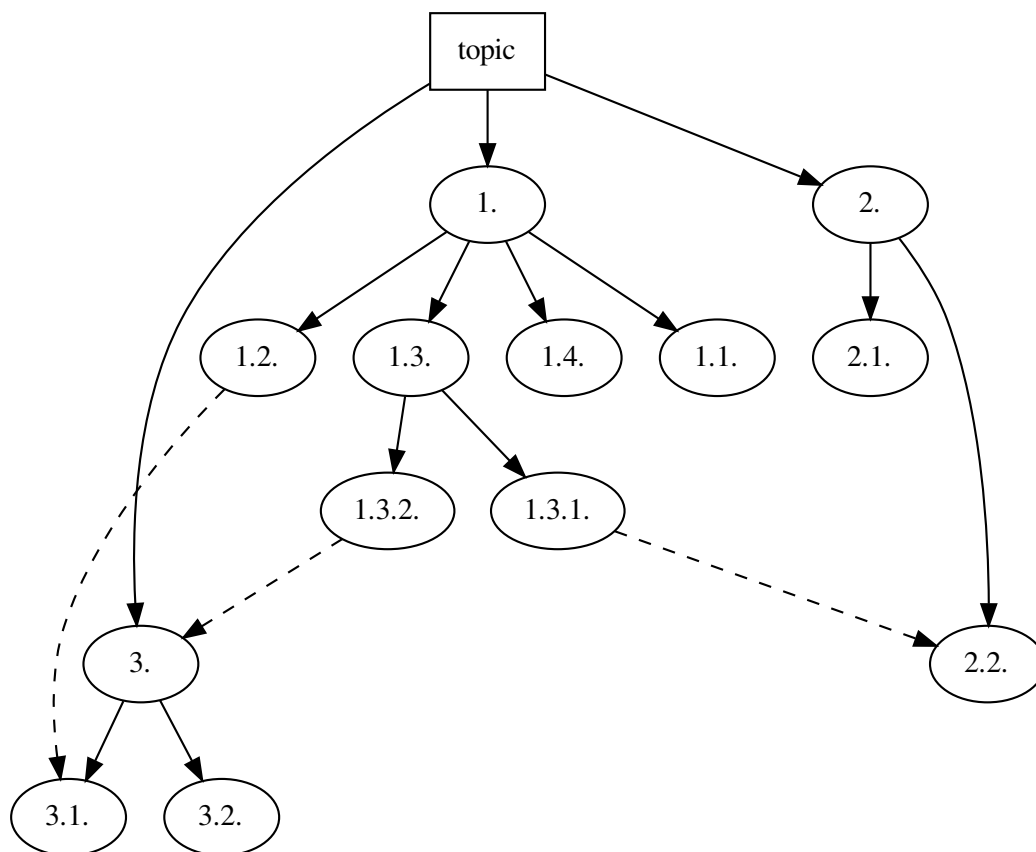
The organization of knowledge in books is equivalent to an ordered [depth-first search](#).



which results in a document outline of:

- 1.
- 1.1.
- 1.2.
- 1.3.
- 1.3.1.
- 1.3.2.
- 1.4.
- 2.
- 2.1.
- 2.2.
- 3.
- 3.1.
- 3.2.

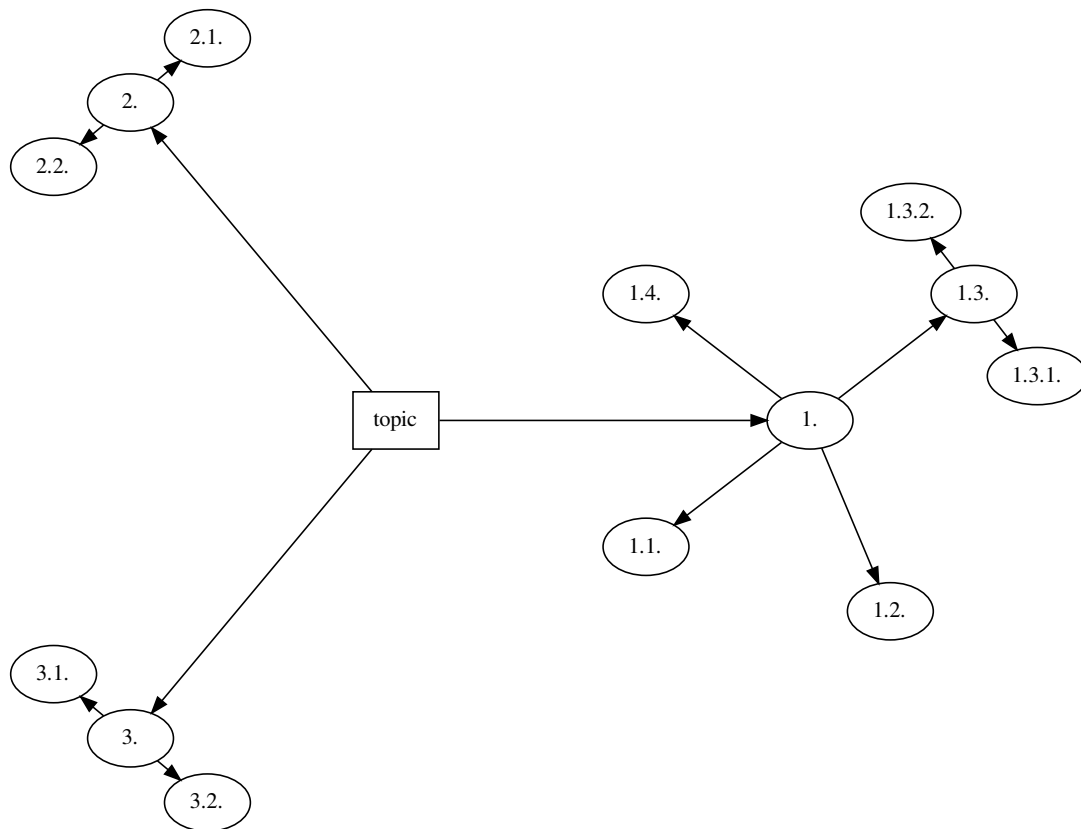
Additional paths, which violate the tree conditions can be introduced with cross-references:



21.1.4 Mindmaps

The popular mindmap also follows a tree structure. For a quick introduction see the online services [MindMup](#) (can store on Google Drive), [Mindmap erstellen kostenlos - ohne Anmeldung](#). The application [freemind\(1\)](#) (/srv/install/freemind, java application) is quite popular with mindmap fans. Mindmaps also allow to connect nodes directly, just like books with their cross-references.

The fact, that a mindmap is nothing but a simple tree, presented in a different manner can be visualized by using a circular layout engine for the book example:



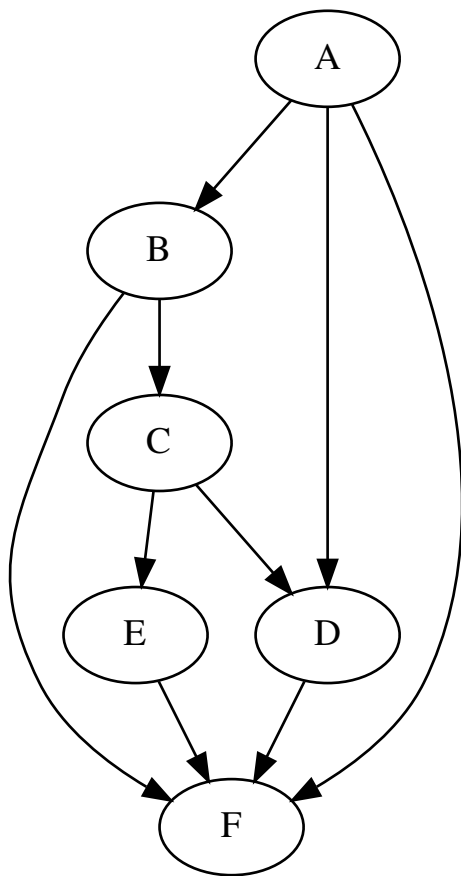
21.2 Generalization

Given an example with 6 sections, where each section shares a subset of information with at least 2 other sections and at most 3 other sections:

```

A -> B;
A -> F;
A -> D;
B -> C;
B -> F;
C -> D;
C -> E;
D -> F;
E -> F;
  
```

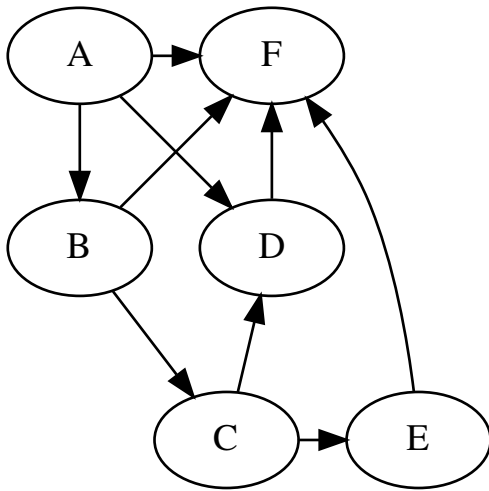
graphviz dot(1) draws an acceptable graph, which is quite tree like:



When some restrictions are introduced, which request certain sections to be of equal importance:

```
subgraph { rank = same; A; F; }  
subgraph { rank = same; B; D; }  
subgraph { rank = same; C; E; }
```

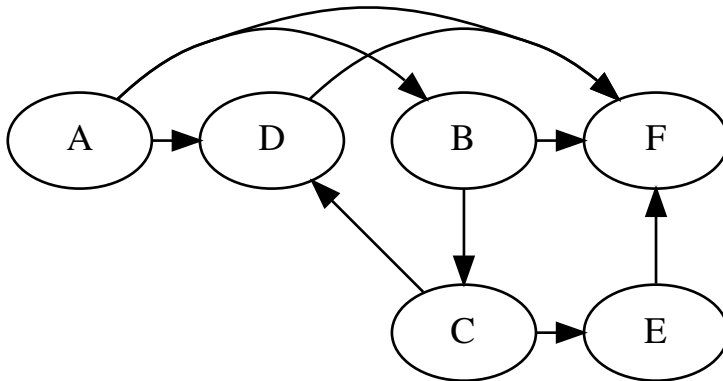
the graph looks less like a tree and more like a graph:



With one additional restriction:

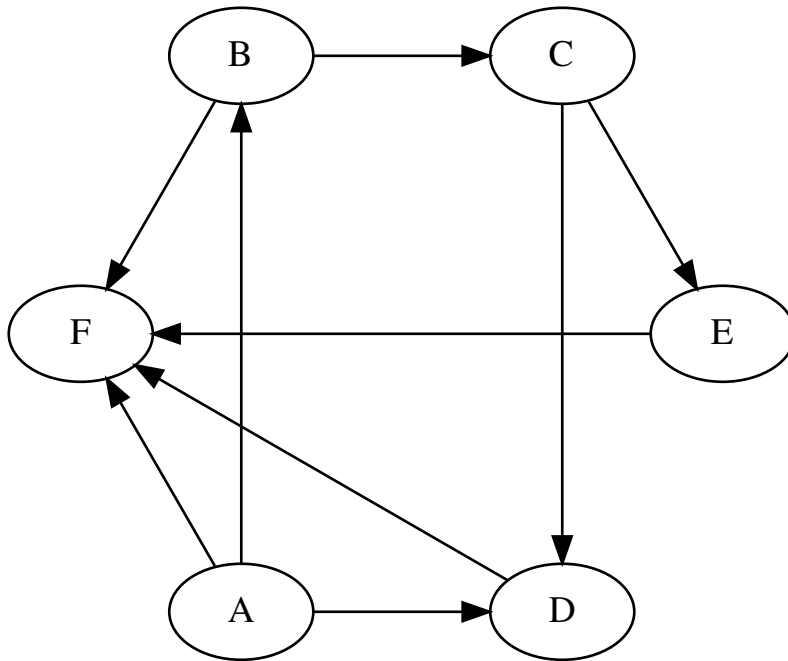
```
subgraph { rank = same; A; F; }
subgraph { rank = same; B; D; }
subgraph { rank = same; C; E; }
subgraph { rank = same; B; F; }
```

the faint resemblance of a tree is gone:

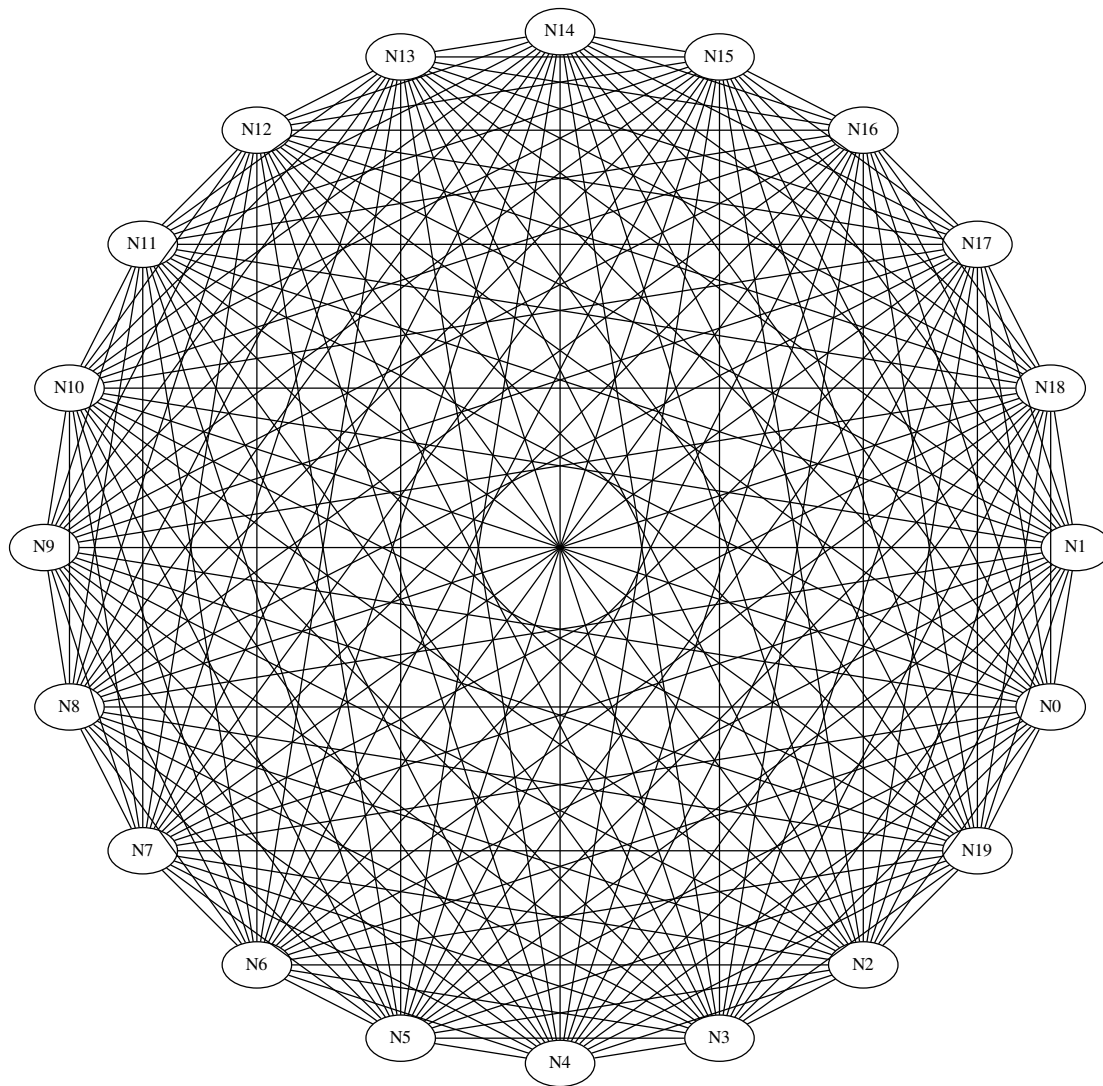


21.2.1 Circular Graphs

Giving up on trees, the most obvious layout to accommodate arbitrary graphs with any structure is a circular one:



The graph for the very simple example might still convey some interesting structural information. The general case, however, really does not help much in visualizing structural relations, although it does visualize the exponentiality of all possible pathes quite nicely (see [Total number of Spanning Trees in a Graph](#)):



21.2.2 Venn Diagrams

Represent the simple example as a Venn diagram:

```

<script src="https://d3js.org/d3.v4.min.js"></script>
<script src="http://benfred.github.io/venn.js/examples/./venn.js"></script>
<script>
// define sets and set set intersections
var sets = [
  {sets: ['A'], size: 12},
  {sets: ['B'], size: 12},
  {sets: ['C'], size: 12},
  {sets: ['D'], size: 12},
  {sets: ['E'], size: 12},
  {sets: ['F'], size: 12},
  {sets: ['A','B'], size: 2},
  {sets: ['A','F'], size: 2},
  {sets: ['A','D'], size: 2},
  {sets: ['B','C'], size: 2},
  {sets: ['B','F'], size: 2},
  {sets: ['C','D'], size: 2},
  {sets: ['C','E'], size: 2},

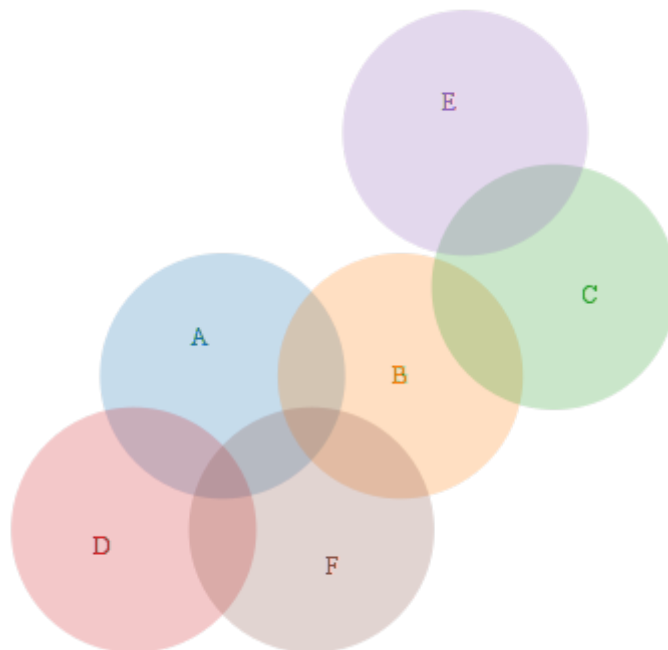
```

(continues on next page)

(continued from previous page)

```
{sets: ['D', 'F'], size: 2},  
  {sets: ['E', 'F'], size: 2}  
  ];  
var chart = venn.VennDiagram();  
d3.select("#venn").datum(sets).call(chart);  
</script>
```

the algorithm for Venn diagrams implemented in [benfred/venn.js](#): Area proportional Venn and Euler diagrams in JavaScript renders nice graphics:



However, it cannot draw all of the common subsections:

WARNING: area C,D not represented on screen venn.js:1737:17

WARNING: area E,F not represented on screen venn.js:1737:17

This makes Venn diagrams practically useless for representing complex matters.

21.3 Conclusion

The fallacy of trees is the assumption that knowledge does indeed have some magical structure. This assumption does of course belong to the set of *unfounded brain-dead assumptions*.

The example of a language reference, which is essentially a dictionary, shows that there is only a superficially useful categorization possible. Beyond that, the possibilities of constructing programs (sentences) are far too numerous to be captured in a book.

The same is true for any attempt of describing good practices or tips and tricks. This includes the current chapter as well as the entire documentation treatise.

Therefore, most mapping efforts trying to impose a rigorous tree structure are usually futile.

As for programming, this means that “object oriented” is not the ultimate answer. Especially things like Java, C#, strong typing in general are far too inflexible for interconnected designs. This is where Lisp, Python, Javascript are exponentially better suited for the task.

Summarily, the collection of knowledge in books is better than nothing, but with the possibilities of digital information storage it appears very antiquated. It is indeed really hard to `grep(1)` through a paper book.

SCRATCH

Miscellaneous stuff to be ordered and processed later.

22.1 Style Guide

Despite the capitalization rules for book titles, e.g. as described in [Capitalisation - The University of Nottingham](#):

Books, films, songs, games etc

Capitalise the first word of the title and all words within the title except articles (*a/an/the*), prepositions (*to/on/for* etc) and conjunctions (*but/and/or* etc).

[...]

Subtitles

Capitalise subtitles only if the original title is printed or displayed that way.

It is simpler to just do away with title capitalization altogether, not just for subtitles.

However, if title capitalization is desired but the correct way is uncertain, enter the title all lowercase into the [Title Case Converter – A Smart Title Capitalization Tool](#) and select *Wikipedia* style.

22.2 Emacs

22.2.1 Symbol tags

Delimiter set concept:

```
[ ANCHOR-RX ] LEFT-DLM SYMBOL-RX RIGHT-DLM [ TRAILER-RX ]
```

Standard delimiter set:

```
| : sym : |
```

Alternate delimiter set:

```
:: sym ::
```

Enclose delimiter set:

```
` sym `
```

Navigation:

key sequence	description
M-left, M-right	previous, next tag
M-up, M-down	previous, next <i>here</i> tag
M-h	go to next <i>here</i> tag. If not found, go to previous <i>here</i> tag
C-u M-<up>	define <i>here</i> tag and go to previous <i>here</i> tag

key sequence	description
M-o	tag occur
M-g	tag grep-find

key sequence	description
M-i	M-x symbol-tag-insert
C-u M-h	insert <i>here</i> tag on line of its own
M-e	enclose current symbol in enclose delimiters

key sequence	description
M-keypad++	more bars for tag at point
M-keypad--	less bars for tag at point
M-C++	more bars for future inserted tags
M-C--	less bars for future inserted tags

key sequence	description
C-c C-d C-v	delimiter stack
f8	delimiter selection menu

22.3 Other Diagrams

- Flowchart (replaced by UML Activity Diagrams)
- Specification and Description Language (SDL)

22.4 Activity Diagrams

PlantUML has no built-in support for the rake symbol ($\perp\!\!\!\perp$) which designates call behavior in UML 2.x. `_static/call-behavior.puml` contains the macros RAKE and CALL, which use UNICODE block characters to provide it.

```

/' |:here:| '/
!include settings.puml
!include call-behavior.puml

partition "Activities with call behavior" {

    fork
    /* Embedded CALL macro invocation with 1 argument */
    :argument of
macro invocation
CALL(is the longest line)
followed by
shorter lines;

    /* Alternative CALL macro with 2 arguments */

```

(continues on next page)

(continued from previous page)

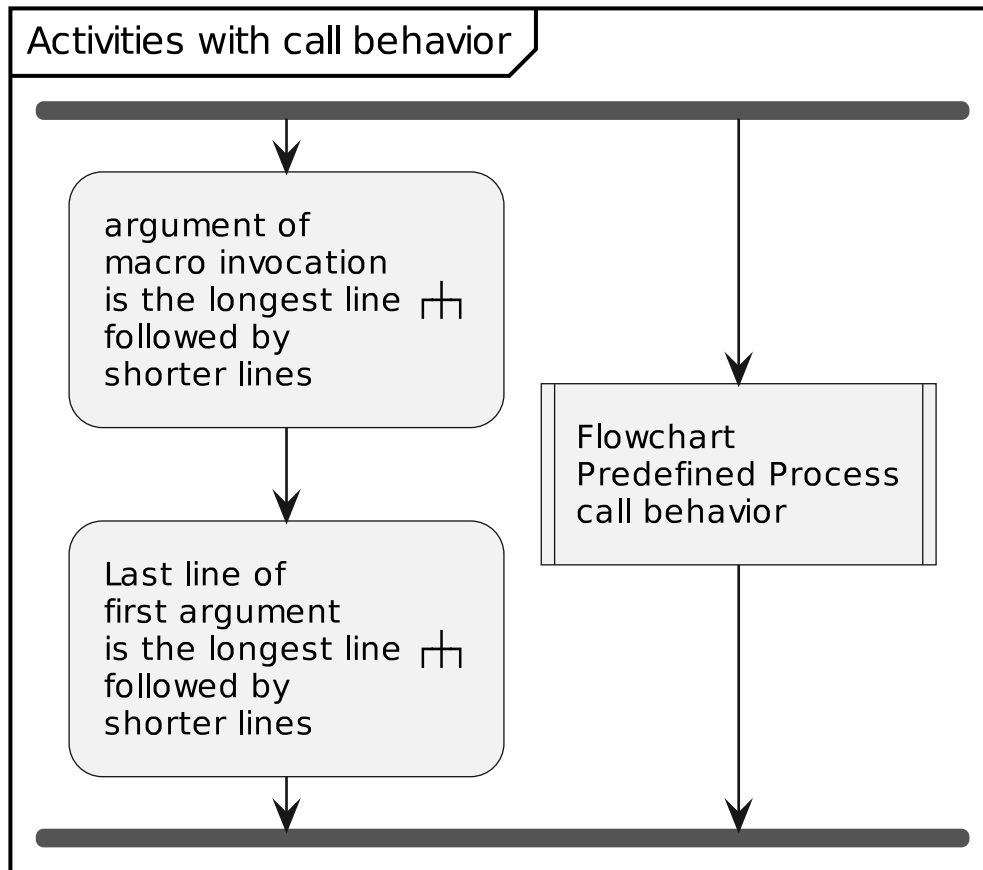
```

:CALL>Last line of\nfirst argument\nis the longest line, \nfollowed by\nshorter lines);

fork again
:Flowchart\nPredefined Process\ncall behavior|

end fork
}

```



22.5 Emacs Buffers with Highlighting

The command `M-x htmlify-buffer` generates HTML source from the current emacs buffer.

The program `webvector` converts HTML to SVG as vector graphics (not bitmap images). `convert(1)` is used to convert SVG to PDF.

Here is an example of the OCCUR-OUTLINE for a python buffer:

host	proces- sors	bo- gomips	mem- ory	real	user	sys	load1	load5	load15
luna.wiedenmann.intern	1	7182.70	10G	13m33.56s	6m55.069s	0m25.234s			
luna.wiedenmann.intern	4	7182.70	9G	4m12.369s	7m8.329s	0m40.473s	2.00	1.45	0.69
sheckley.simul.de	8	6983.85	62G	5m1.218s	7m26.416s	0m29.068s	3.24	2.77	1.91
goch.wiedenmann.intern	4	6624.16	31G	4m19.433s	7m40.044s	0m23.544s	2.01	1.31	0.74
scherer.wiedenmann.intern	1	6000.00	31G	3m26.640s	8m19.654s	0m28.548s	3.11	1.60	0.76
tilman-Vostro-5568	4	5424.00	7G	4m47.822s	13m4.366s	0m32.663s	2.60	1.86	1.09
spectre.simul.de	8	3984.00	15G	4m11.647s	13m59.743s	0m38.552s	2.62	1.90	1.02

Note:

real < user => process uses multiple CPU's

real > user => process loses time for I/O or virtualization

Theoretically *luna* (Virtualbox VM) with a single processor loses a lot of time (real == 13m33, user == 6m55) to virtualization.

Since the build process obviously uses more than 1 CPU for building diagrams, increasing the CPUs to 4 results in a substantial gain for a full build. However, an incremental build without diagrams is not faster with 4 CPUs compared to a single CPU.

A further speed-up, but probably marginal, can be expected if Linux is running on the bare metal and windows is running in the virtual machine.

QUESTIONS

1. Does it make a difference how i write the HTML-Colour-Code (#ffcccc vs. #01DF01)?

There is no difference except lower/uppercase in your examples. Both are hexadecimal specifications of RGB colors.

2. You show me some things with Sphinx, but what are the tasks for me to do with Sphinx.

Sphinx is used to write documentation.

3. Is `gen_plantuml.py` a python snippet for creating a fast UML Diagram?

No it is a script, but it does not produce UML class diagrams the way they must be.

1. if yes, should I use it?

No, not now. Yes, for other modules.

1. if yes, how?

As it is intended (see output of command `gen_plantuml.py` for usage).

4. What is a programming language pre-processor

Look up C-preprocessor. The pre-processor does not generate target code (assembler) but transforms the source code into another version of source code.

- substitute constants, e.g. `#define CONSTANT 23456`
- expand macros, e.g. `#define MACRO(x, y) (x + y)`
- include files, e.g. `#include "xyz.h"`
- keep or remove source depending on conditions, e.g. `#if defined(abc) ... #endif`
- Remove comments

5. Enum == normally a list of all variables in the whole programm?

No, an enum is exactly the same as an enum in C/C++:

```
enum { xyz = 2, abc, def=5, ghi };

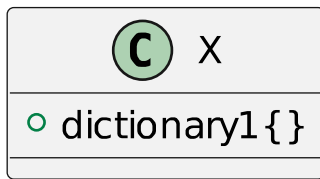
const int xyz = 2;
const int abc = 3;
const int def = 5;
const int ghi = 6;
```

6. If I have multiple classes, should I write for every class class "different_class_names" as C ?

No.

- "Class 0" as C0
- "Class 1" as C1
- Class2

7. Should I write {} in the UML for die type description of a dictionary, even though, I should use a other Syntax for Dictionaries?



Do not confuse modeling with an actual language syntax!

8. I've never used private attributes and methods in python (I know, I told me what is going in with private things in pyhon, but I forgot it. I only know, that _private tells an other programmer, that it is private and SHOULD NOT be changed, because if you do, you could collapse the whole thing.

That is all there is to know.

1. Should I use private attributes and methods in python?

1. If yes, when and how?

Yes, if you do not wish to guarantee an API.

2. If no, should I though use the private declarations in the UML to describe the programm, in case, some one will the UML to write the Programm in a programming language in which you use privates?

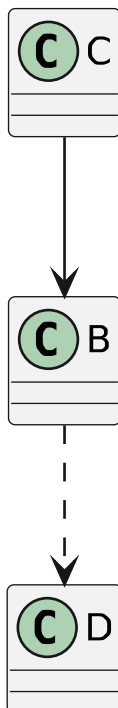
9. Do I need more than E, C and G for describing every thinkable programm?

Probably.

10. Is the name of a method/function/attribute/variable etc. enough, or should/could I also add a small commend to describe it?

As you please.

11. Quote: "It is possible to replace – by .. for broken lines". What does broken lines means in this context?



12. Is Directed Association like two telefones, where one only can transmit voice and the other only can receive it?

Good question. But no, it is not a strictly unidirectional thing.

13. What's about Dependency, Generalization/Extension, Realization?

Wikipedia.

14. Syntax question: In your first example you wrote class "ClassDiagram" as C and in your Diagramm appeared a C. In your BB example, you wrote ... as BB, but no BB appeared in your diagramm. Why?

15. Why is the ws-cartoon-logo a puml type?

Check !include statement in Reference Guide.

16. Is the logo also saved in the _static directory?

Check TXT + Files.

17. Why is who[] and who_not[] from type list and not from type dict?

Because they are references to data, stored in other places.

It implies that there is a dictionary somewhere. But it should be a global registry of all persons (usu. a database).

who and *who_not* are categories and should not contain the entire person object, just an ID (name or whatever).

There is not even a need for a persistent list, it might as well be the temporary result of a database query:

```
SELECT * FROM PERSONS WHERE status = "suspicious"
```

18. I assume the text after the methods and the : is the short description a asked some questions above. Is that right?

Feel free.

19. Would you get a Syntax Error, if you would'nt use the // in your long descriptions (for instance after suspect(p))

No, since // is Creole wiki syntax for italics, as used by PlantUML.

20. Is the uml directive only visual in HTML output?

No, it is also in PDF, EPUB is just a HTML archive, so it is obviously there.

21. What does BB circle do?

The command starts at the beginning of the line:

```
hide BB circle
```

It hides the circle (with the C) for the class *BB*.

22. Why does everybody use Foo and Bar for examples????

FUBR -> Fucked Up Beyond Recognition (military slang)

23. M-x plantuml-preview == *Ctrl-c u u v* in your emacs?

Not quite, but almost. Help for key: *Ctrl-h k Ctrl-c u u v*:

```
C-c u u v runs the command x-plantuml-preview-current-block, which is an interactive Lisp function in `x-plantuml.el'.
```

```
Find next UML diagram and call `plantuml-preview-current-block'.
```

24. Should I also handle yUML although it's not easy to use it offline?

No, it is just listed as evaluated alternative. The need to be online disqualifies it. As does the limited amount of diagrams and the bad readability of the diagram definitions.

25. Do I need to handle the Other UML Tools (GUI with UML standart support, Generic Diagrams ...)
No. I have just evaluated them to give a hint as to which ones are most useful, most up to date.

RST-MODE ETAGS SUPPORT

24.1 Resources

- README-rst-mode-etags-support.txt
- bin/diffmap.py
- bin/rst_tags.py
- _static/rst-TAGS linked to TAGS

listing 24.1: workspace register setup

```
(workspace-set-locations '(
  ("/home/sw/project/documentation/README-rst-mode-etags-support.txt" 5122 "README-rst-mode-
  ↪etags-support.txt" 49 nil)
  ("/home/ws/project/documentation/bin/rst_tags.py" 15145 "rst_tags.py" 50 nil)
  ("/home/ws/project/documentation/bin/diffmap.py" 3482 "diffmap.py" 51 nil)
  ("/home/sw/project/documentation/_static/rst-TAGS" 1 "rst-TAGS" 55 nil)
) nil t t)
```

After evaluating the expression in listing 24.1 (C-x C-e at end of expression, or copy expression, M-:, yank expression, RET),

```
+-----+-----+-----+
|       |       |       |
|       |       |       |
| 7 _____ | 8 | 9 |
|       |       |       |
|       |       |       |
| 4 _____ | 5 | 6 |
|       |       |       |
|       |       | 3 _____ |
|       | 2 _____ |
| 1 _____ |
|       |
| 0 |
|       |
+-----+
```

• /home/sw/project/documentation/_static/rst-TAGS

• /home/ws/project/documentation/bin/diffmap.py

• /home/ws/project/documentation/bin/rst_tags.py

• /home/sw/project/documentation/README-rst-mode-etags-support.txt

24.2 etags Interface

A special include entry is useful to combine several files:

```
^L
documentation/TAGS,include
^L
administration/TAGS,include
^L
```

(continues on next page)

(continued from previous page)

```
../administration/README-redistribution-of-info-items.txt,30
.. _`sec:Info tags`^?sec:Info tags^A370,11867
```

The TAGS file with special characters represented as octal escape sequences in brackets:

```
<\014>
README-rst-mode-etags-support.txt,57
.. _`sec:Sample TAGS File`:<\177>sec:Sample TAGS File<\001>21,6337
<\014>
../administration/README-redistribution-of-info-items.txt,30
.. _`sec:Info tags`<\177>sec:Info tags<\001>370,11867
```

24.3 diffmap

diffmap.py - generate line mapping from target to source from diff(1) output

usage:	diffmap.py [OPTIONS] source-or-diff-output [target]
or	import diffmap

24.3.1 Options

-q, --quiet	suppress warnings
-v, --verbose	verbose test output
-d, --debug	show debug information
-h, --help	display this help message
-t, --test	run doc tests

24.3.2 Description

The diff(1) between two source and transformed file gives an accurate specification which line numbers of the two files are correlated.

```
diff -u /home/ws/project/administration/README-redistribution-of-info-items.txt /home/ws/
↔project/administration/doc/redistribution-of-info-items.rst.auto
```

```
@@ -1,7 +1,18 @@
```

The hunk description specifies, that

- the block of lines in source document consisting of 7 lines starting at line 1, matches with
- the block of lines in target document consisting of 18 lines starting at line 1

see [text processing - How shall I understand the unified format of diff output? - Unix & Linux Stack Exchange](#).

The correlation can then be determined from the hunk information alone, without parsing the input files:

Note: The mapping is different, depending on a hunk having some lines or no lines!

```
>>> text = '\n'.join(('line ' + str(_i) for _i in xrange(15)))
```

s_from	t_from
1	1 → 1
2	2 → 2
3	3 → 3
4	4 → 4

s_from	t_from
1	1 → 1
2	2 → 2
	3 → 2
	4 → 2

s_from	t_from
1	1 → 1
2	2 → 2
3	
4	

figure 24.1: Correlate Line Numbers from Diff Output


```
>>> def write_file(file, line_nos):
...     with open(file, 'w') as _fh:
...         lines = list(text.splitlines())
...         for _i in line_nos:
...             printf(lines[_i], file=_fh)
```

```
>>> def dump_file(file):
...     printf('\n'.join([sformat("{0:02d} | {1}", _i + 1, _l)
...                       for _i, _l in
...                           enumerate(open(file, 'r').read().splitlines())]))
```

Line mapping, when source file hunk is completely removed

```
>>> write_file("t1", [1, 2, 3, 7, 8])
>>> write_file("t2", [1, 2, 3, 4, 5, 6, 7, 8])
```

```
01 | line 1  <-----  01 | line 1
02 | line 2  <-----  02 | line 2
03 | line 3  <-----  03 | line 3
                        \----- 04 | line 4  \  These are between line 03
                        \---- 05 | line 5  >  > and line 04 of source file
                        \-- 06 | line 6  /
04 | line 7  <-----  07 | line 7
05 | line 8  <-----  08 | line 8
```

```
@@ -3,0 +4,3 @@
+line 4
+line 5
+line 6
```

Lines 00 - 06 of target file map to lines 00 - 03 of source file. Therefore, lines 04 - 06 all map to line 03. Lines starting at 7 are mapped with an offset of -3 (07 -> 04, 08 -> 05).

```
>>> dm = DiffMap("t1", "t2")
>>> print(dm)
# :INF: s_line      : ]4[
# :INF: t_line      : ]7[
# :INF: s_line - t_line: ]-3[
# 0 - 6 ( 3) => 0 - 3
```

Line mapping, when target file hunk is completely removed

```
>>> write_file("t1", [1, 2, 3, 4, 5, 6, 7, 8])
>>> write_file("t2", [1, 2, 3, 7, 8])
```

```
01 | line 1  <-----  01 | line 1
02 | line 2  <-----  02 | line 2
03 | line 3  <-----  03 | line 3
04 | line 4  <---/
05 | line 5  <--/
06 | line 6  <-/
07 | line 7  <-----  04 | line 7
08 | line 8  <-----  05 | line 8
                        \----- 04 | line 4  \  These are between line 03
                        <---/
                        <--/
                        <-/
                        /
                        >  > and line 04 of target file
```

```
--- t1...
+++ t2...
@@ -4,3 +3,0 @@
-line 4
-line 5
-line 6
```

Lines 00 - 03 of target file map to lines 00 - 03 of source file. Lines starting at 04 are mapped with an offset of +3 (04 -> 07, 05 -> 08).

```
>>> dm = DiffMap("t1", "t2")
>>> print(dm)
# :INF: s_line : ]7[
# :INF: t_line : ]4[
# :INF: s_line - t_line: ]3[
# 0 - 3 ( 3) => 0 - 3
```

Line mapping, when source and target file hunk overlap

```
>>> write_file("t1", [1, 2, 3, 4, 5, 6, 7, 8])
>>> write_file("t2", [1, 2, 3, 3, 3, 7, 8])
```

```
01 | line 1 <----- 01 | line 1
02 | line 2 <----- 02 | line 2
03 | line 3 <----- 03 | line 3
04 | line 4 <----- 04 | line 3 \ These lines overlap between
05 | line 5 <----- 05 | line 3 > source and target file
06 | line 6 <-/ /
07 | line 7 <----- 06 | line 7
08 | line 8 <----- 07 | line 8
```

```
--- t1...
+++ t2...
@@ -4,3 +4,2 @@
-line 4
-line 5
-line 6
+line 3
+line 3
```

Lines 00 - 05 of target file map to lines 00 - 05 of source file. Lines starting at 06 are mapped with an offset of +1 (06 -> 07, 07 -> 08).

```
>>> dm = DiffMap("t1", "t2")
>>> print(dm)
# :INF: s_line : ]7[
# :INF: t_line : ]6[
# :INF: s_line - t_line: ]1[
# 0 - 5 ( 5) => 0 - 5
```

```
>>> os.unlink('t1')
>>> os.unlink('t2')
```

24.3.3 Module

24.3.4 Automatic Exports

```
>>> for ex in __all__: printf(sformat('from {0} import {1}', __name__, ex))
from diffmap import DiffMap
```

24.3.5 Explicit Exports

```
>>> if '__all_internal__' in globals():
...     for ex in __all_internal__:
...         printf(sformat('from {0} import {1}', __name__, ex))
```

24.3.6 Details

class `diffmap.DiffMap` (*source_or_diff_output=None, target=None*)

C DiffMap
<ul style="list-style-type: none"> ○ <code>s_line</code> : integer ○ <code>t_line</code> : integer ○ <code>t_to_s_map</code> : dict
<ul style="list-style-type: none"> ● <code>__str__()</code> ● <code>ranges()</code> ● <code>parse_diff(diff_output)</code>
<ul style="list-style-type: none"> ● <code>parse(source_file, target_file)</code> ● <code>lookup(lookup_t_line) : mapped_s_line</code>

figure 24.2: Correlate Line Numbers from Diff Output

```
>>> _debug = sys.modules["__main__"]._debug
```

```
>>> text = '\n'.join(('line ' + str(_i) for _i in xrange(15)))
```

```
>>> def write_file(file, line_nos):
...     with open(file, 'w') as _fh:
...         lines = list(text.splitlines())
...         for _i in line_nos:
...             printf(lines[_i], file=_fh)
```

```
>>> def dump_file(file):
...     printf('\n'.join([sformat("{0:02d} | {1}", _i + 1, _l)
...                         for _i, _l in
...                         enumerate(open(file, 'r').read().splitlines())]))
```

```
>>> write_file('/tmp/diffmap-source.txt', list(xrange(15))[1:])
>>> write_file('/tmp/diffmap-target.txt', [1, 2, 3, 4, 4, 5, 5, 6, 7, 11, 12, 13])
```

```
>>> dump_file('/tmp/diffmap-source.txt') #doctest: +ELLIPSIS
01 | line 1
02 | line 2
03 | line 3
04 | line 4
05 | line 5
06 | line 6
07 | line 7
08 | line 8
09 | line 9
10 | line 10
11 | line 11
12 | line 12
13 | line 13
14 | line 14
```

```
>>> dump_file('/tmp/diffmap-target.txt') #doctest: +ELLIPSIS
01 | line 1
02 | line 2
```

(continues on next page)

(continued from previous page)

```

03 | line 3
04 | line 4
05 | line 4
06 | line 5
07 | line 5
08 | line 6
09 | line 7
10 | line 11
11 | line 12
12 | line 13

```

```

>>> printf(os.popen("diff --unified=0 '/tmp/diffmap-source.txt' '/tmp/diffmap-target.txt'
↔", 'r').read(), end='') #doctest: +ELLIPSIS
--- /tmp/diffmap-source.txt...
+++ /tmp/diffmap-target.txt...
@@ -4,0 +5,2 @@
+line 4
+line 5
@@ -8,3 +9,0 @@
-line 8
-line 9
-line 10
@@ -14 +12,0 @@
-line 14

```

```

>>> dm = DiffMap('/tmp/diffmap-source.txt', '/tmp/diffmap-target.txt')
>>> printf(dm)
#      :INF:      s_line      : ]15[
#      :INF:      t_line      : ]13[
#      :INF:      s_line - t_line: ]2[
#  0 -    6 (  4) =>    0 -    4
#  7 -    9 (  9) =>    5 -    7
# 10 -   12 ( 12) =>   11 -   13

```

```

>>> for _item in ditems(dm):
...     printf(_item)
(0, 0)
(1, 1)
(2, 2)
(3, 3)
(4, 4)
(5, 4)
(6, 4)
(7, 5)
(8, 6)
(9, 7)
(10, 11)
(11, 12)
(12, 13)

```

```

>>> os.unlink('/tmp/diffmap-source.txt')
>>> os.unlink('/tmp/diffmap-target.txt')

```

clear()

Clear data.

lookup(*t_line*)

Lookup target line.

Returns mapped source line for target line.**parse(*source_file*, *target_file*)**

Diff source and target file, parse diff output.

parse_diff(*diff_output*)

Parse hunks in diff output.

listing 24.2: check for diff edge case empty file

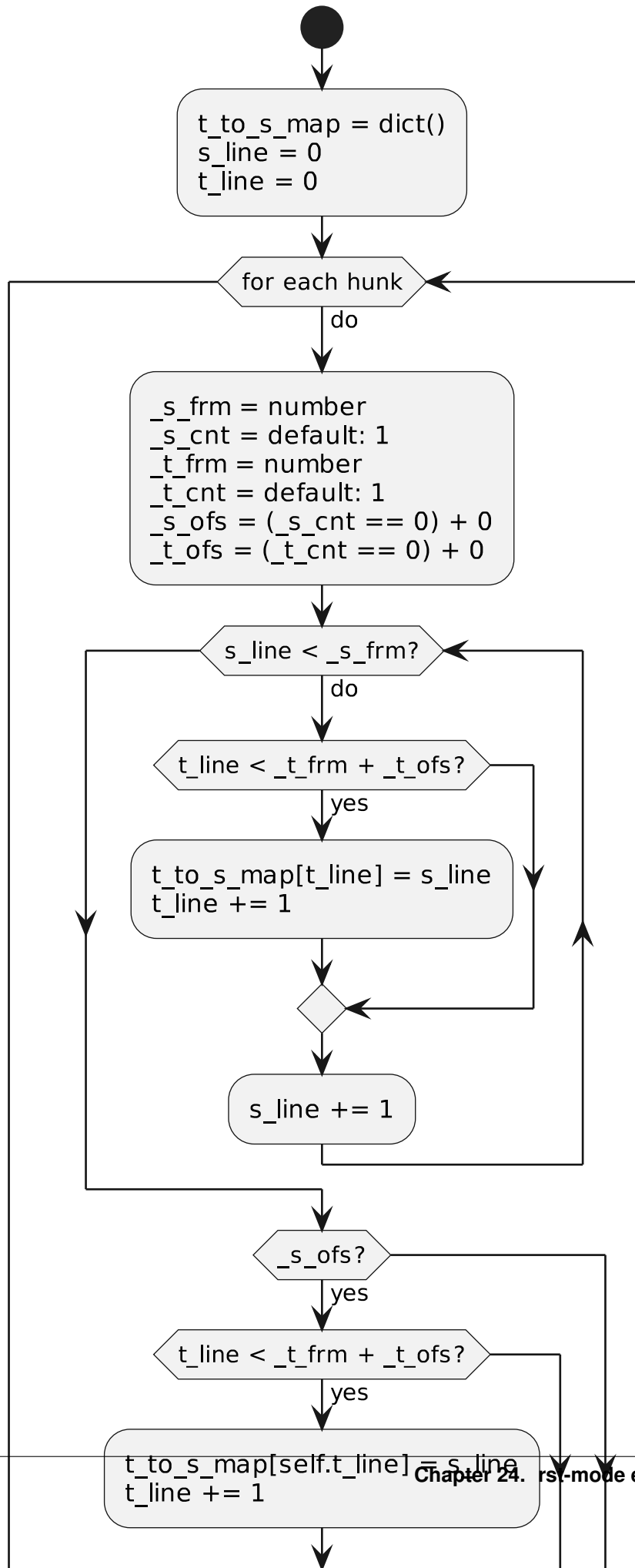
```
printf "%s\n%s\n%s\n" 'hallo' 'hallo' 'hallo' | diff -u /dev/null -
```

listing 24.3: check for diff edge case empty file, diff output

```
--- /dev/null      2022-10-29 11:10:30.081667219 +0200
+++ -             2023-03-08 17:53:12.495842805 +0100
@@ -0,0 +1,3 @@
+hallo
+hallo
+hallo
```

ranges ()

Returns list of mapped ranges



ABBREVIATIONS

abbr see *Abbreviation*

EASE Eclipse Advanced Scripting Environment

HTML see *HyperText Markup Language*

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

JPEG see *Joint Photographic Experts Group*

PDF Portable Document Format

PNG see *Portable Network Graphics*

REST Representational State Transfer

RPC Remote Procedure Call

SDL see *Specification and Description Language*

SGML Standard Generalized Markup Language

SVG Scalable Vector Graphics

XML see *Extensible Markup Language*

GLOSSARY

Abbreviation As Wikipedia describes it[WPABBR]:

An abbreviation (from Latin *brevis*, meaning short) is a shortened form of a word or phrase. It consists of a group of letters taken from the word or phrase. For example, the word abbreviation can itself be represented by the abbreviation *abbr.*, *abbrv.*, or *abbrev.*

Extensible Markup Language An *SGML* application like *HTML*.

HyperText Markup Language An *SGML* application.

Joint Photographic Experts Group Norm ISO/IEC 10918-1, image file format

Portable Network Graphics An image file format (.PNG)

Specification and Description Language A standardized language, described by Wikipedia[WPSDL]:

As of 2016 its current areas of application include process control and real-time applications in general. Due to its nature it can be used to represent simulation systems without ambiguity and with a graphical notation.

BIBLIOGRAPHY

- [DICTTERM] Dictionary.com, LLC. Term, Definition of Term at Dictionary.com. Retrieved 12:52, May 27, 2019, from <https://www.dictionary.com/browse/term>
- [WPSDL] Wikipedia contributors. (2019, March 26). Specification and Description Language. In Wikipedia, The Free Encyclopedia. Retrieved 15:46, August 25, 2019, from https://en.wikipedia.org/w/index.php?title=Specification_and_Description_Language&oldid=889575411
- [WPABBR] Wikipedia contributors. (2019, May 9). Abbreviation. In Wikipedia, The Free Encyclopedia. Retrieved 15:58, May 21, 2019, from <https://en.wikipedia.org/w/index.php?title=Abbreviation&oldid=896285984>
- [WPTERM] Wikipedia contributors. (2019, May 15). Terminology. In Wikipedia, The Free Encyclopedia. Retrieved 12:52, May 27, 2019, from <https://en.wikipedia.org/w/index.php?title=Terminology&oldid=897146677>
- [TORC] Torczyner, Harry. Magritte: Ideas and Images. p. 71. ISBN 978-0-8109-1300-4.

PYTHON MODULE INDEX

d

diffmap, 189

l

line_diversion, 45

s

sphinx_doc_snip, 132

A

a black blade runner, **30**
 abbr, **197**
 Abbreviation, **198**
 abrr, **30**
 assemble() (line_diversion.LineParser method), 51

C

check_line_parse() (in module line_diversion), 49
 clear() (diffmap.DiffMap method), 194
 close_partition() (line_diversion.Diagram method), 52
 close_partition() (line_diversion.LineDiversion method), 53

D

Diagram (class in line_diversion), 52
 DiagramActivity (class in line_diversion), 52
 DiagramBehavior (class in line_diversion), 52
 DiagramClass (class in line_diversion), 52
 DiagramComponent (class in line_diversion), 52
 DiagramDeployment (class in line_diversion), 52
 DiagramInteraction (class in line_diversion), 52
 DiagramMessageSequence (class in line_diversion), 52
 DiagramObject (class in line_diversion), 52
 DiagramStateMachine (class in line_diversion), 52
 DiagramStructure (class in line_diversion), 52
 DiagramTiming (class in line_diversion), 52
 DiagramUseCase (class in line_diversion), 52
 DiffMap (class in diffmap), 193
 diffmap (module), 189
 Document (class in sphinx_doc_snip), 135
 Documentation (class in sphinx_doc_snip), 134
 dump() (sphinx_doc_snip.Snippet method), 143
 dump_snippets() (sphinx_doc_snip.Document method), 143
 dump_snippets() (sphinx_doc_snip.Documentation method), 135

E

EASE, **197**
 Extensible Markup Language, **198**

F

first, **30**
 finish() (line_diversion.LineDiversion method), 53
 first, **30**

H

HTML, **197**
 HTTP, **197**
 HyperText Markup Language, **198**

I

IDE, **197**
 init() (line_diversion.LineParser method), 51

J

Joint Photographic Experts Group, **198**
 JPEG, **197**

L

line_diversion (module), 45
 LineDiversion (class in line_diversion), 53
 LineParser (class in line_diversion), 50
 load() (sphinx_doc_snip.Document method), 143
 lookup() (diffmap.DiffMap method), 194

M

map_snippets() (sphinx_doc_snip.Document method), 143
 match() (line_diversion.LineParser method), 51

O

open_partition() (line_diversion.Diagram method), 52
 open_partition() (line_diversion.LineDiversion method), 53

P

parse() (diffmap.DiffMap method), 194
 parse() (sphinx_doc_snip.Document method), 143
 parse_diff() (diffmap.DiffMap method), 194
 PDF, **197**
 PNG, **197**
 Portable Network Graphics, **198**
 prepare() (sphinx_doc_snip.Snippet method), 143
 process_parts() (line_diversion.Diagram method), 52

R

ranges() (diffmap.DiffMap method), 195
 register() (sphinx_doc_snip.Documentation method), 135
 register_snippets_with() (sphinx_doc_snip.Document method), 143

reset() (sphinx_doc_snip.Snippet method), 143
resolve() (sphinx_doc_snip.Document method), 143
resolve() (sphinx_doc_snip.Documentation method),
135
resolve() (sphinx_doc_snip.Section method), 143
REST, **197**
RPC, **197**
rx (line_diversion.LineParser attribute), 51

S

SDL, **197**
Section (class in sphinx_doc_snip), 143
SGML, **197**
Snippet (class in sphinx_doc_snip), 143
Specification and Description Language, **198**
sphinx_doc_snip (module), 132
split() (line_diversion.LineParser method), 51
split_() (line_diversion.LineParser method), 52
SVG, **197**

X

XML, **197**