Documentation Standard

Release 0.0

Wolfgang Scherer

Dec 17, 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/
                                   # (necesary, 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 counterprodcutive (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 programm 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

LIST OF TABLES

LIST OF CODE BLOCKS

CHAPTER

ONE

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.

CHAPTER

TWO

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 excuted. **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

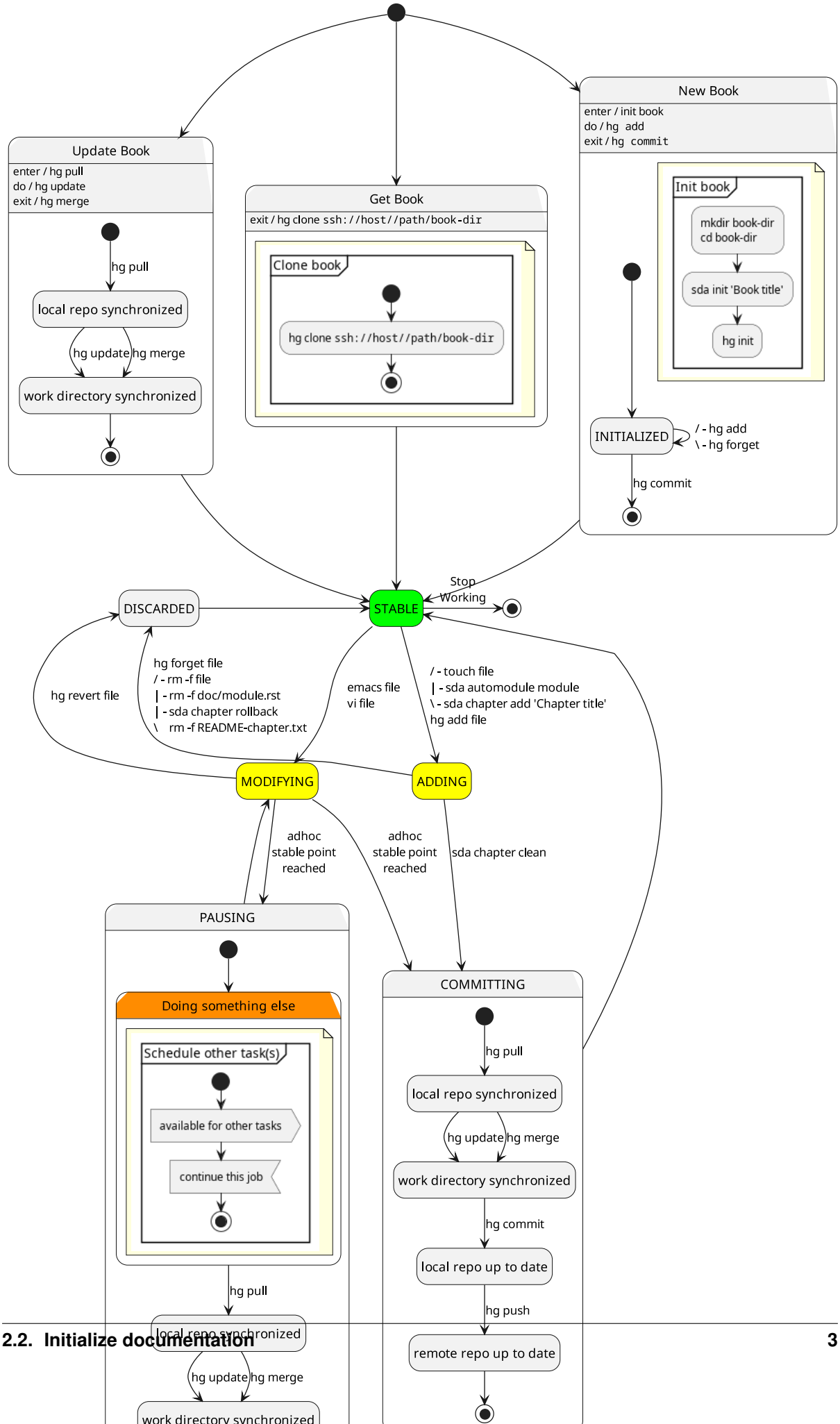
```
snn --mode rst --key title --value 'Chapter title' README-chapter-title.txt
```

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.


Update Book
enter / hg pull
do / hg update
exit / hg merge
hg pull
local repo synchronized
hg update
hg merge
work directory synchronized
Get Book
exit / hg clone ssh://host//path/book-dir
Clone book
hg clone ssh://host//path/book-dir
New Book
enter / init book
do / hg add
exit / hg commit
Init book
mkdir book-dir
cd book-dir
sda init 'Book title'
hg init
INITIALIZED
/ - hg add
\ - hg forget
hg commit
DISCARDED
STABLE
Stop
Working
hg revert file
hg forget file
/ - rm -f file
| - rm -f doc/module.rst
| - sda chapter rollback
\ rm -f README-chapter.txt
emacs file
vi file
/ - touch file
| - sda automodule module
\ - sda chapter add 'Chapter title'
hg add file
MODIFYING
ADDING
adhoc
stable point
reached
adhoc
stable point
reached
sda chapter clean
PAUSING
Doing something else
Schedule other task(s)
available for other tasks
continue this job
hg pull
local repo synchronized
hg update
hg merge
work directory synchronized
COMMITTING
hg pull
local repo synchronized
hg update
hg merge
work directory synchronized
hg commit
local repo up to date
hg push
remote repo up to date

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,;1q' )"
```

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.

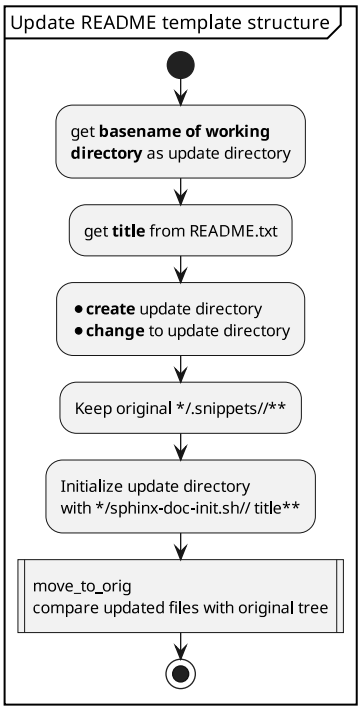


figure 2.2: SphinxDoc - update template structure

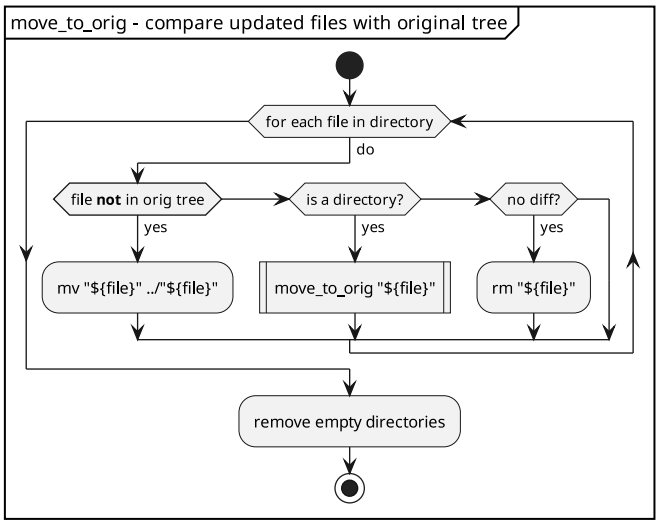
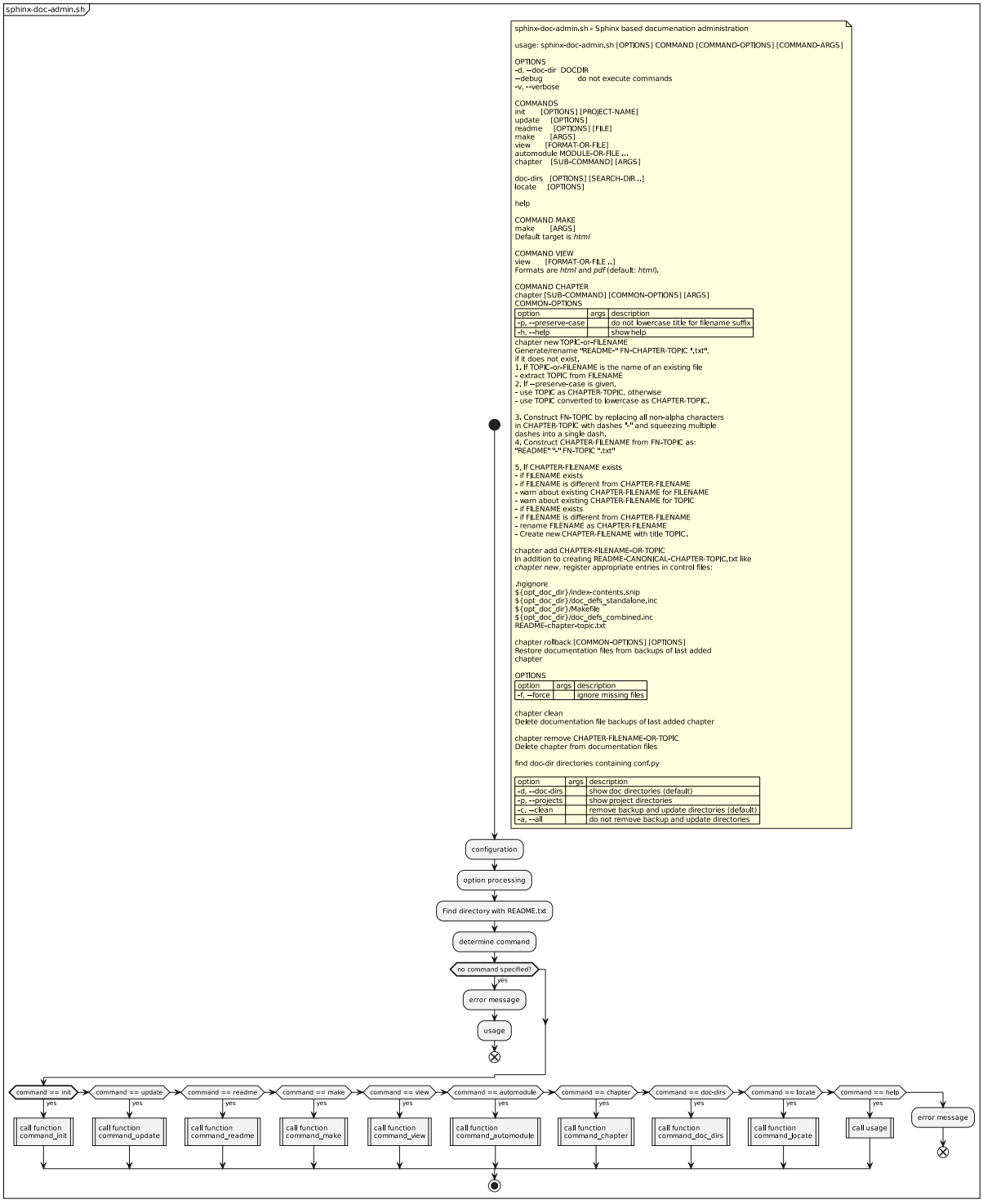


figure 2.3: SphinxDoc - compare updated files with original tree

2.5 Document administration

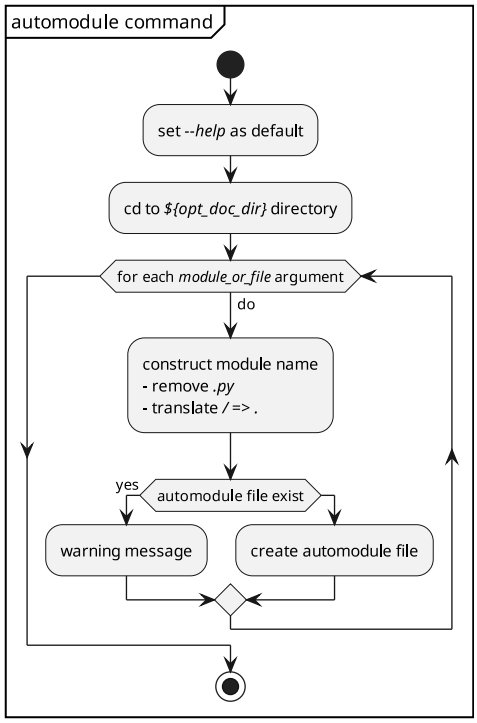
Document administration is performed with **sda**, a shortcut link to **sphinx-doc-admin.sh**.



2.5.1 Automodule files

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

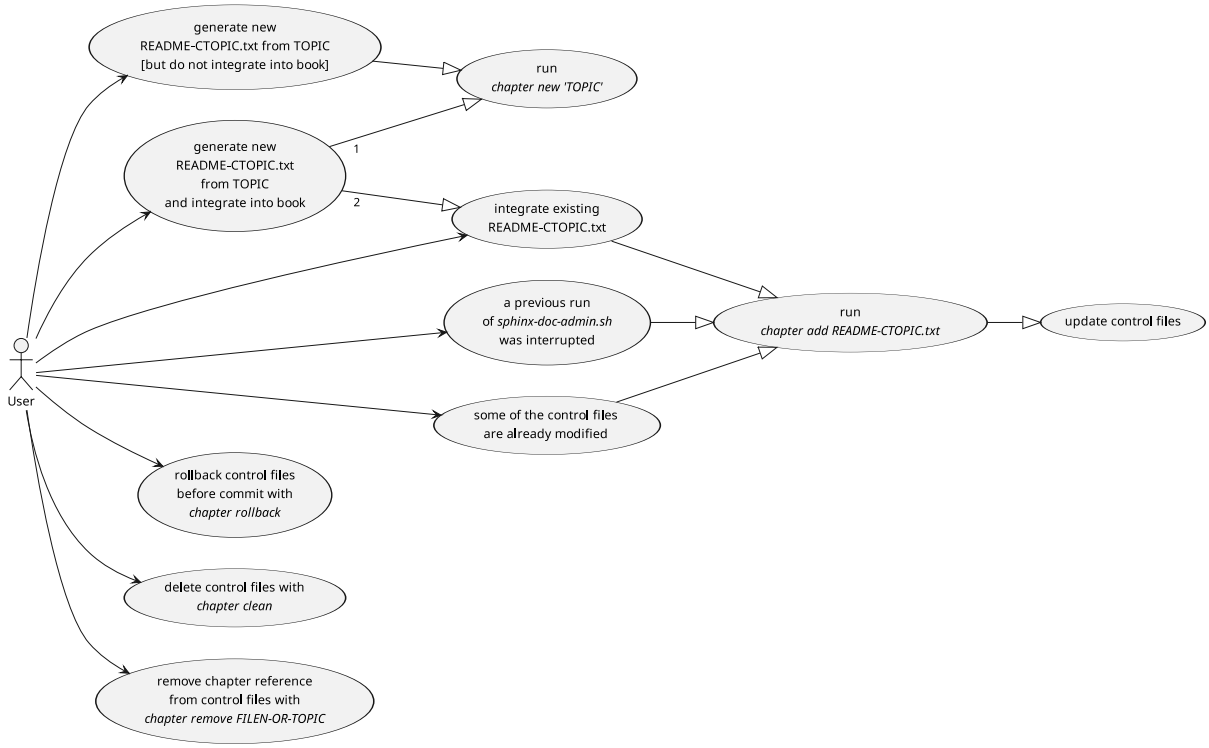
```
sda automodule MODULE-OR-FILE ...
```



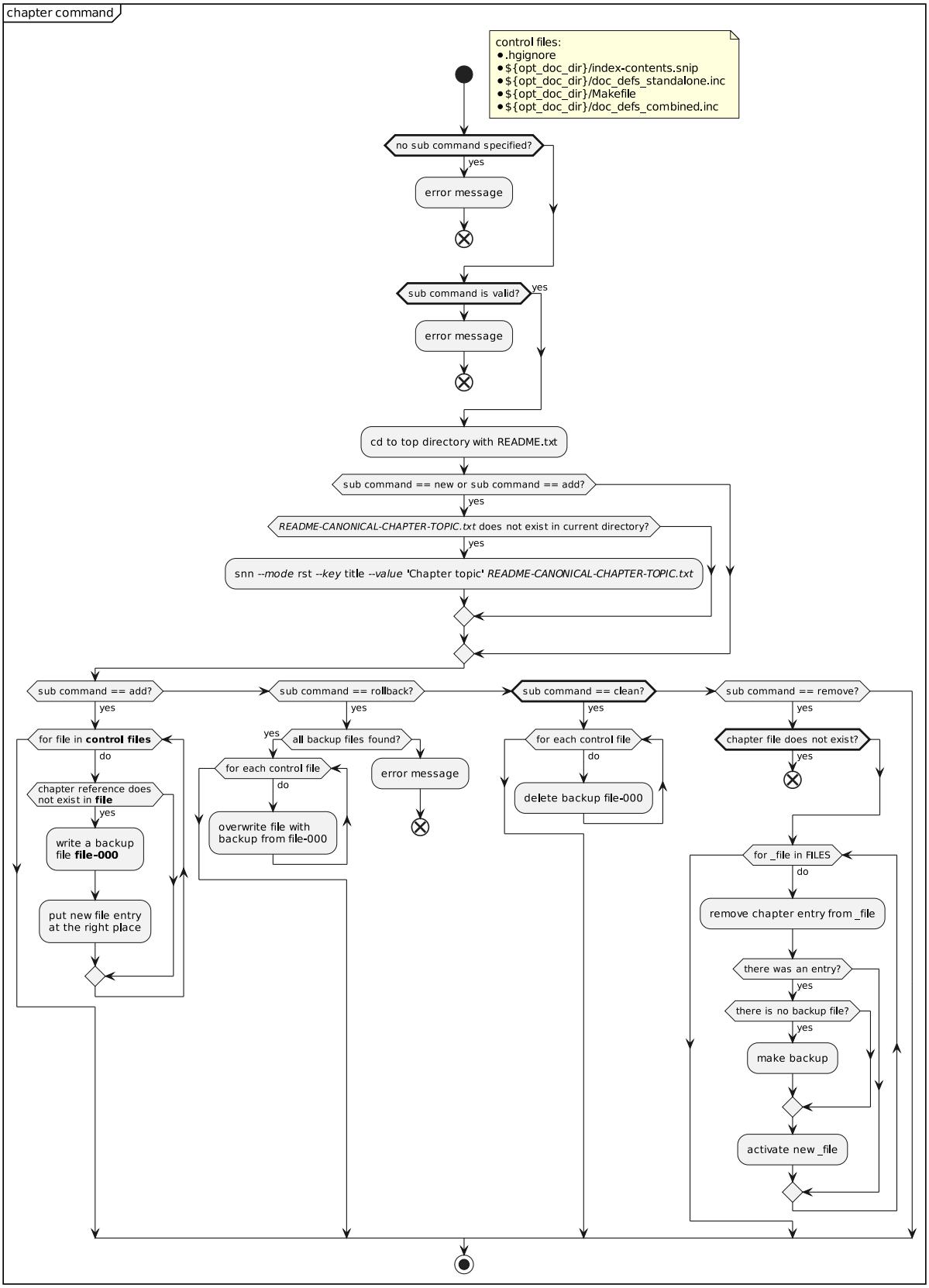
2.5.2 Chapters

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`:

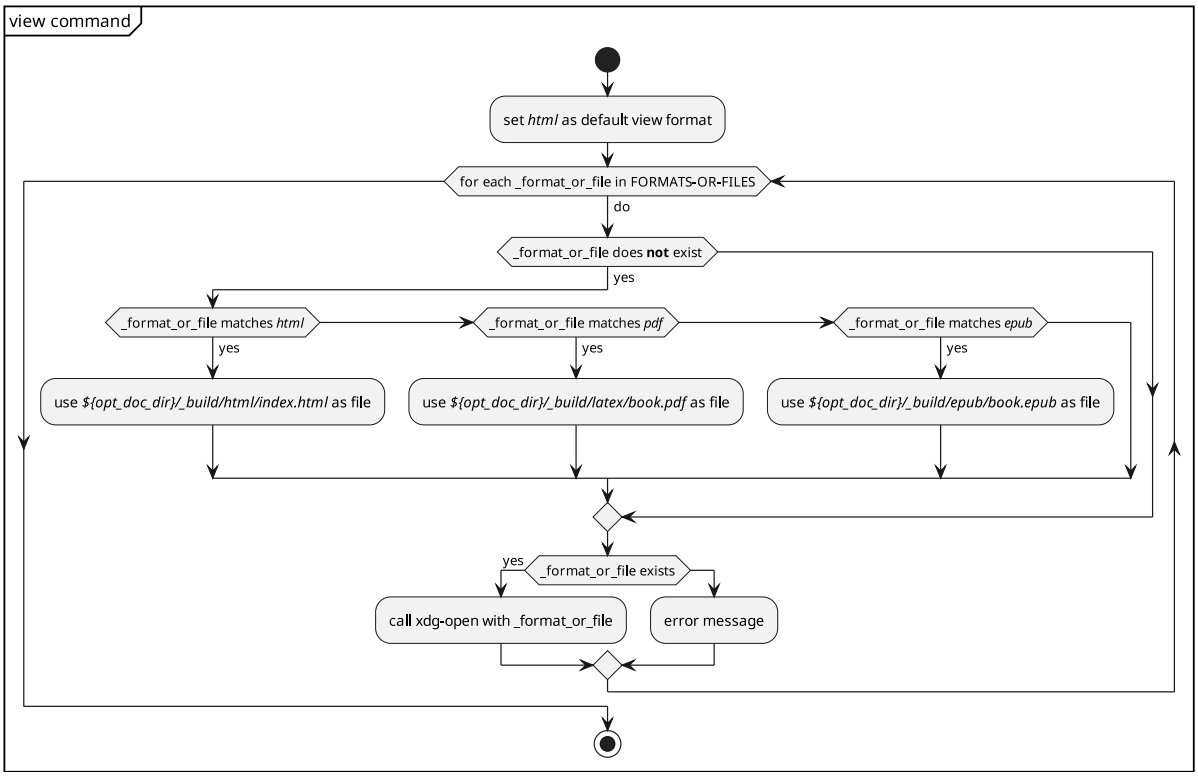
```
.. |chapter-topic|            replace:: document :file:`README-topic`
```

Define a chapter reference in `doc/doc_defs_combined.inc`:

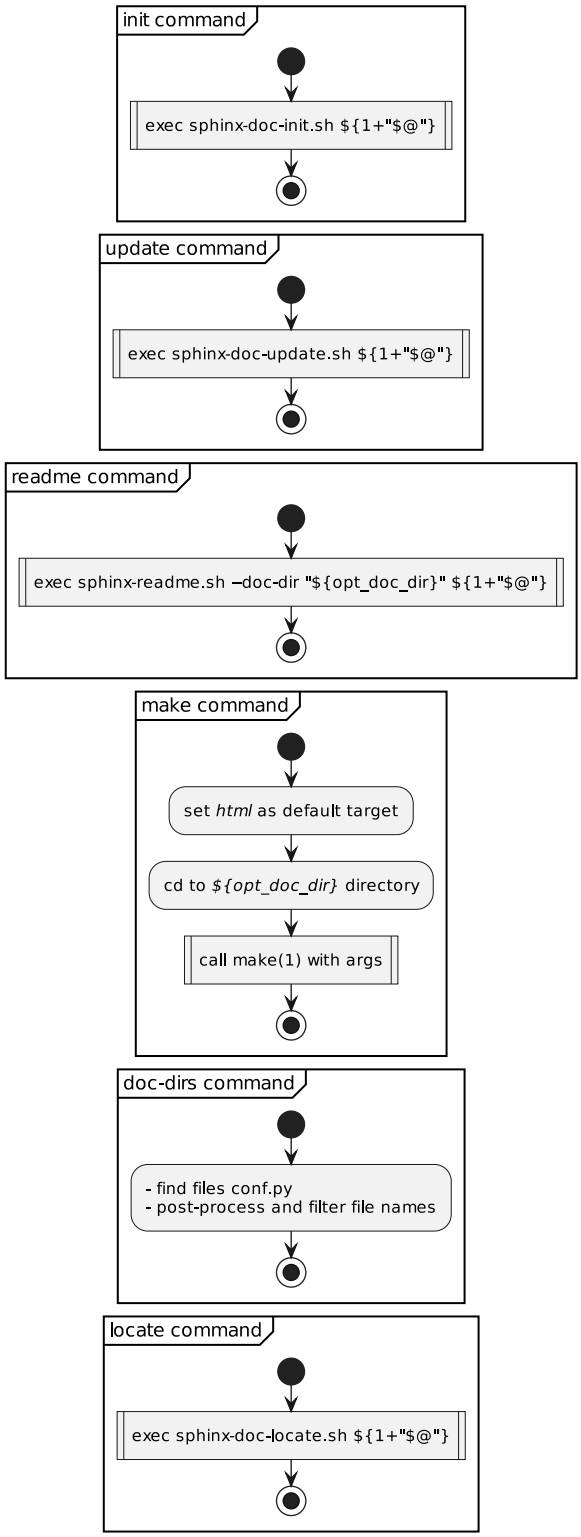
```
.. |chapter-topic|            replace:: chapter :doc:`topic`
```

2.5.3 View processed documents

```
sda view FORMAT-OR-FILE

sda view html
sda view pdf
```



2.5.4 Command handlers for backends

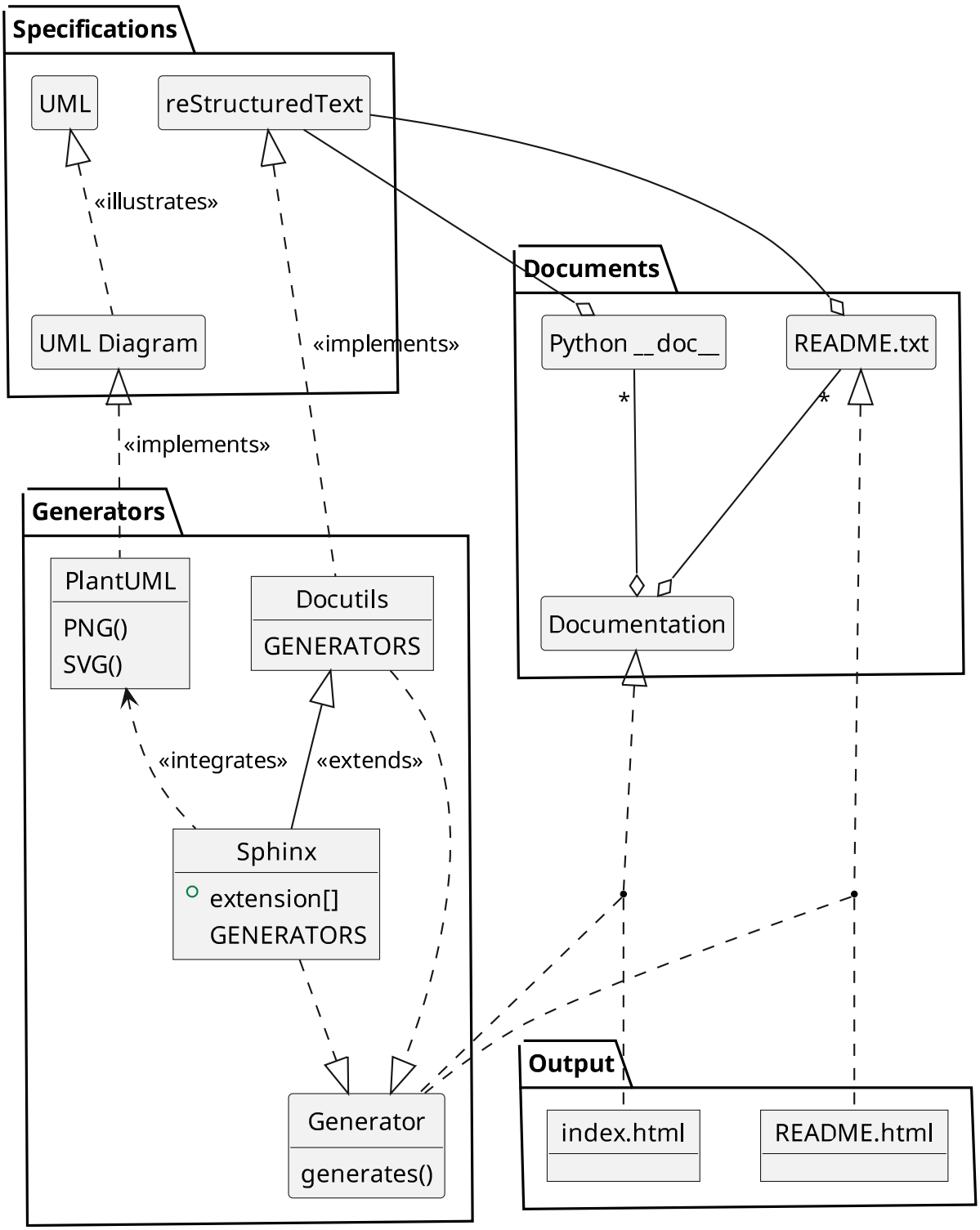


2.6 Structural specification

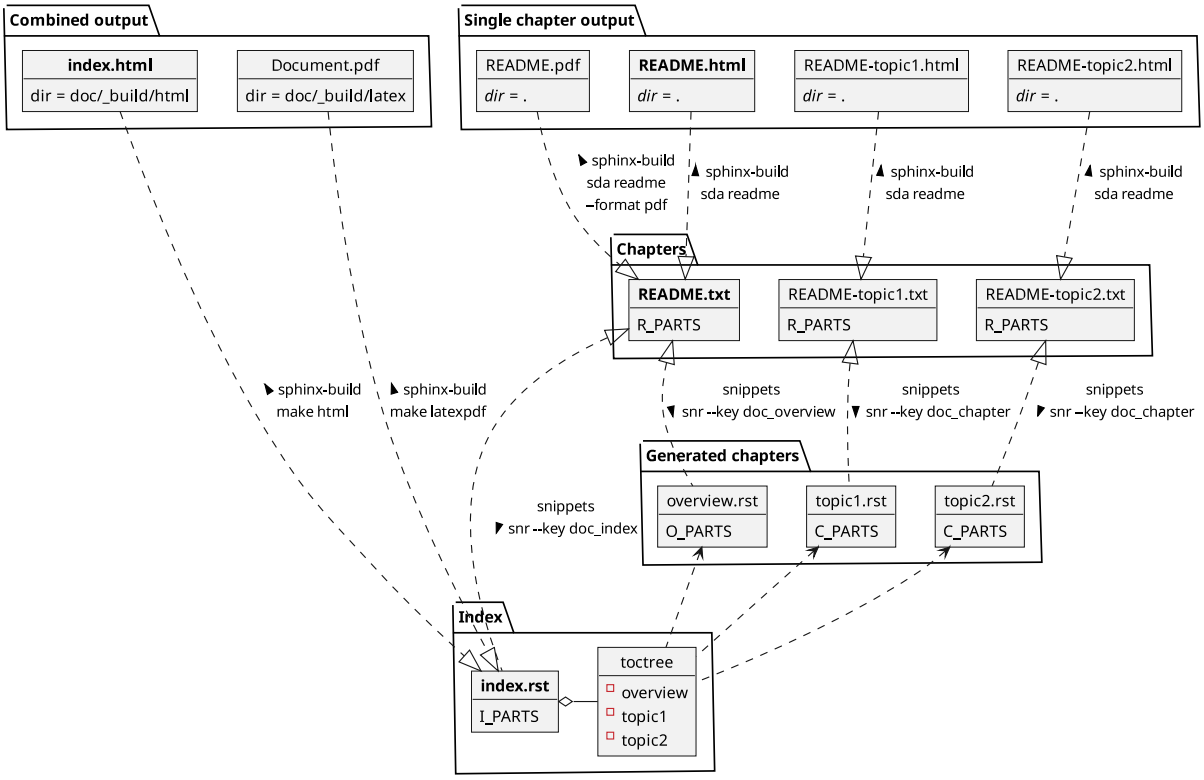
- reStructuredText is a markup language specification in the class of wiki markup languages. (Examples for other markup languages are HTML, TeX/LaTex, NROFF). reStructuredText is the official Python documentation markup.

- 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 .

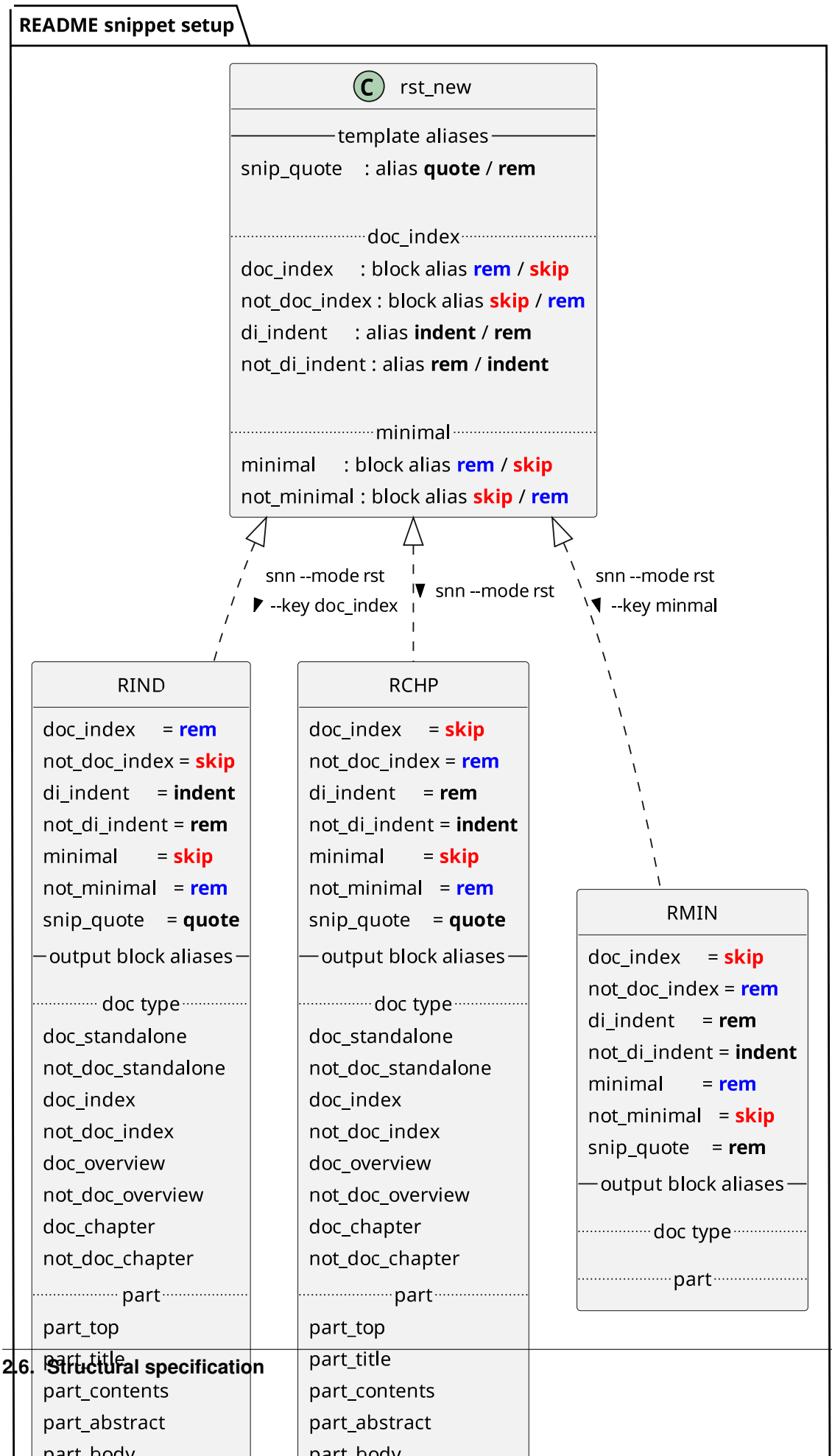
  The program **sphinx-build**(1) generates HTML/PDF/EPUB documentation from reStructuredText documents and doc strings of python(1) modules. It is also used in the command **sda readme** (**sphinx-readme.sh**) to generate standalone HTML/PDF/EPUB from a single `README` file.


Specifications
UML
reStructuredText
«illustrates»
UML Diagram
«implements»
«implements»
Documents
Python __doc__
README.txt
*
*
Documentation
Generators
PlantUML
PNG()
SVG()
Docutils
GENERATORS
«integrates»
«extends»
Sphinx
extension[]
GENERATORS
Generator
generates()
Output
index.html
README.html

2.6.1 README chapters

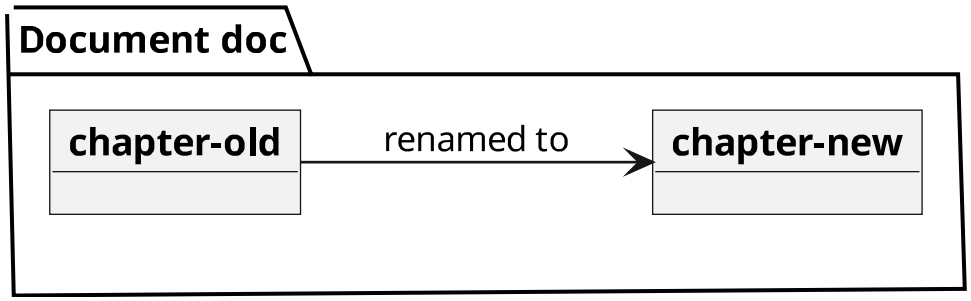


2.6.2 README snippets



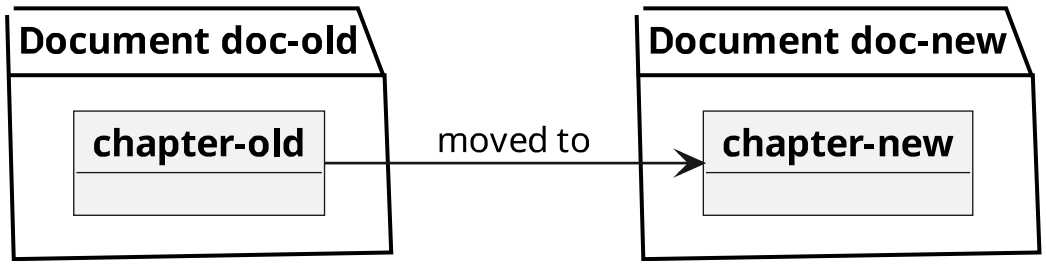
2.7 How to properly move chapter files and sections

2.7.1 Rename chapter file in same document directory



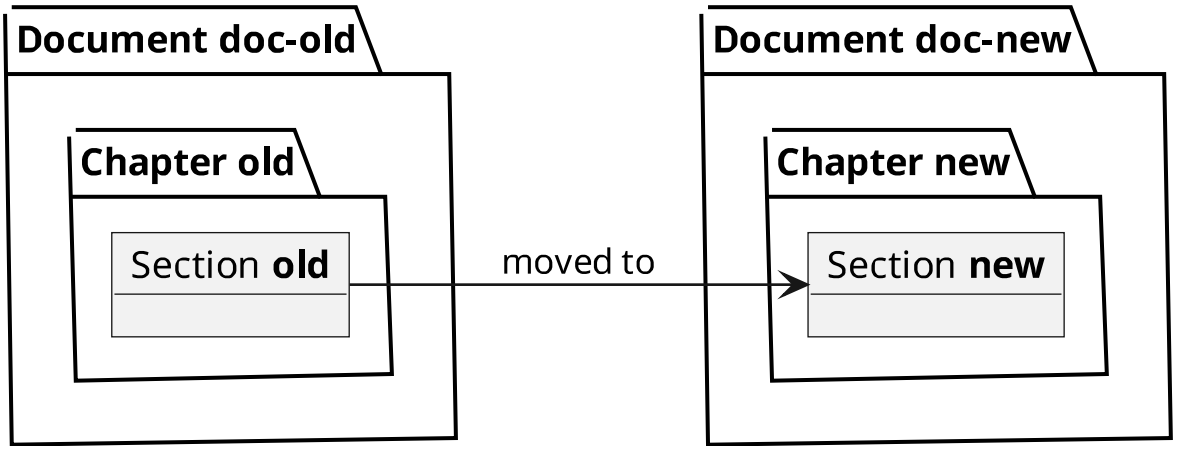
- Move `README-chapter-old.txt` to `README-chapter-new.txt`.
- Execute `M-x grep-find RET chapter-old RET`.
  Rename info for **chapter-old** to info for **chapter-new**.
- Clean + compile **doc-new**.
  Analyze error messages and correct errors.

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	`###`	`===`	`---`	`~~~`	`...`

CHAPTER

THREE

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 appropiate 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)

(continued from previous 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"
```

CHAPTER

FOUR

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`

```
numfig = True
numfig_format = {
    'section': 'section {number}, {name}',
    'figure': 'figure %s',
    'table': 'table %s',
    'code-block': 'listing %s',
}
```

The supported entities are:

- **section** labels.
- Directive **figure** with filename as arguments, first paragraph in body as title, other pragraphs 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

```
.. _`fig:Figure example Magrittes Pipe`:
.. figure:: _static/MagrittePipe.jpg

   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)

```
.. _`tab:single table cell rectangle`:
.. table:: single table cell (rectangle)

   +-----------------------+
   | .. code-block:: text  |
   |                       |
   |    code-block without |
   |    caption inside a   |
   |    single table cell  |
   +-----------------------+
```

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

```
.. _`lst:<LABEL>`:
.. code-block:: <LANGUAGE>
   :caption: <TITLE>
   :linenos:

   <BODY>
```

An example is shown in listing 4.5.

listing 4.5: Example with Elisp code

```
.. _`lst:Some Emacs Lisp code`:
.. code-block:: elisp
   :caption: Some Emacs Lisp code
   :linenos:

   (if (not nil)
      (let (x) (princ x))
     (let(y) princ y))
```

The rendered example is shown in listing 4.6.

listing 4.6: Some Emacs Lisp code

```
(if (not nil)
   (let (x) (princ x))
  (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

```
.. _`fig:UML test`:
.. uml::
   :caption: UML test

   @startuml
   A --> B : send
   A <-- B : receive
   @enduml
```

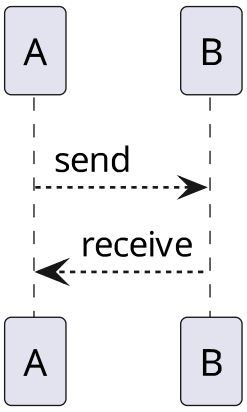


figure 4.3: UML test

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

```
.. _`fig:dot test`:
.. graphviz::
   :caption: dot test

   digraph xx {
   A -> B;
   }
```

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).

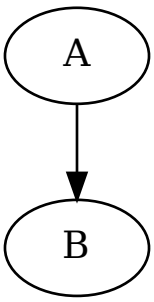


figure 4.4: dot test

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

CHAPTER

FIVE

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.

CHAPTER

SIX

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
Abbreviations **==========** .. **glossary**:: |abbrr| a black blade runner first see :term:`f i r s t` and short :term:`abbrr`	**Abbreviations** ***abbrr*** a black blade runner **first** see *f i r s t* and short *abbrr*
Glossary **========** .. **glossary**:: abbrr a black blade runner some description a black blade runner elaborate entry for |abbrr| f i r s t see :term:`first`	**Glossary** **abbrr** a black blade runner some description **a black blade runner** elaborate entry for *abbrr* **f i r s t** see *first*
.. **|abbrr| replace**:: :term:`abbrr <a black blade runner>`	

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 *abbrr* (`:term:`abbrr``) and an elaborate one *abbrr* (`.. |abbrr| replace:: :term:`tstaddr <a black blade runner>``).

6.3 Combinations of glossary and abbreviation generation

Enable features with option *–enable FLG,FLG,...*:

gl[ossary] 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
	+	+	abbrr see :term:`a black blade runner` a black blade runner some description	abbrr :term:`a black blade runner`	.. |abbrr| replace:: :term:`abbrr <a black blade runner>`
*	+	-	a black blade runner some description	abbrr :term:`a black blade runner`	.. |abbrr| replace:: :term:`abbrr <a black blade runner>`
	-	+	abbrr a black blade runner Some description	|abbrr| a black blade runner	.. |abbrr| replace:: :term:`abbrr`
	-	-		abbrr a black blade runner	.. |abbrr| replace:: :term:`abbrr`

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

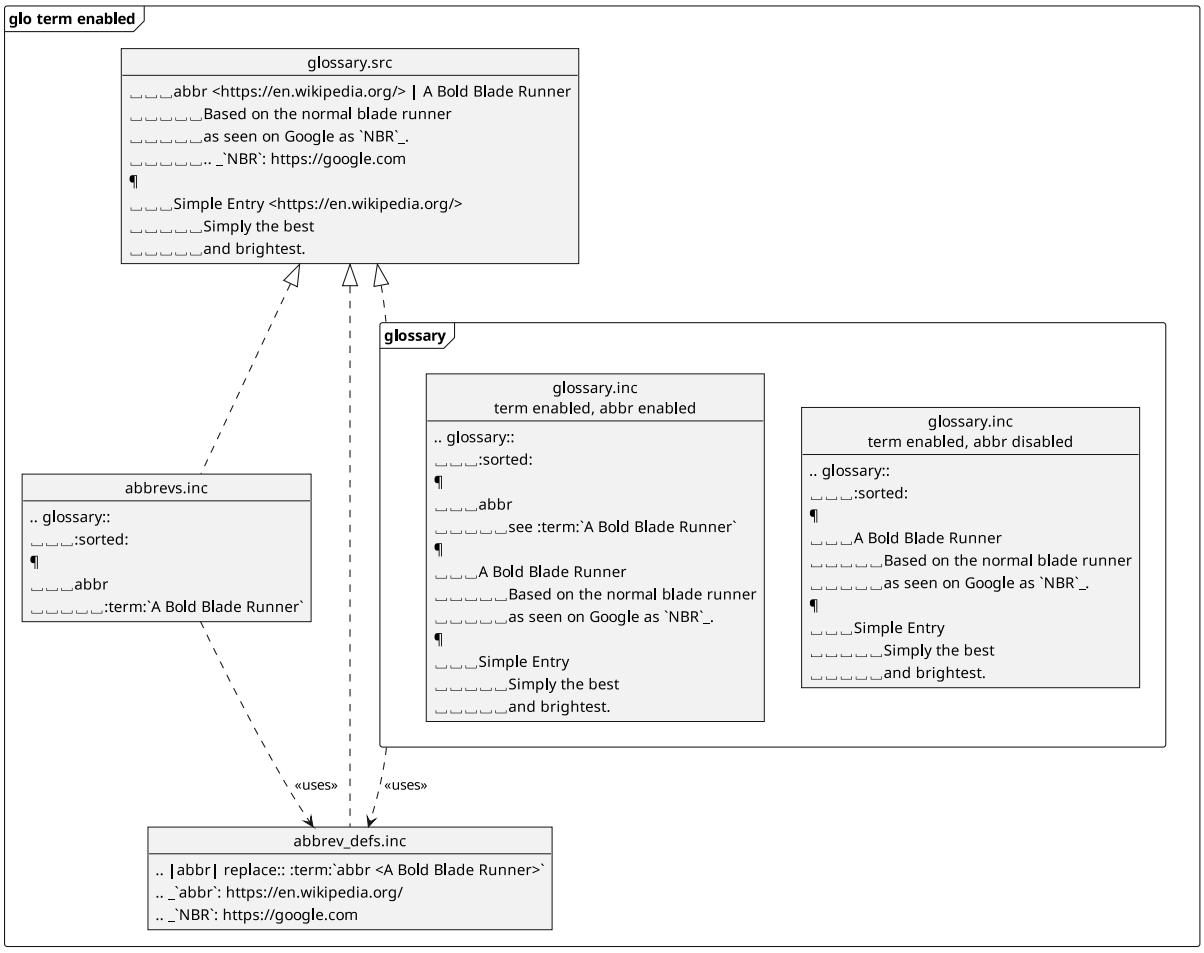


figure 6.2: Generated output files for glossary terms enabled

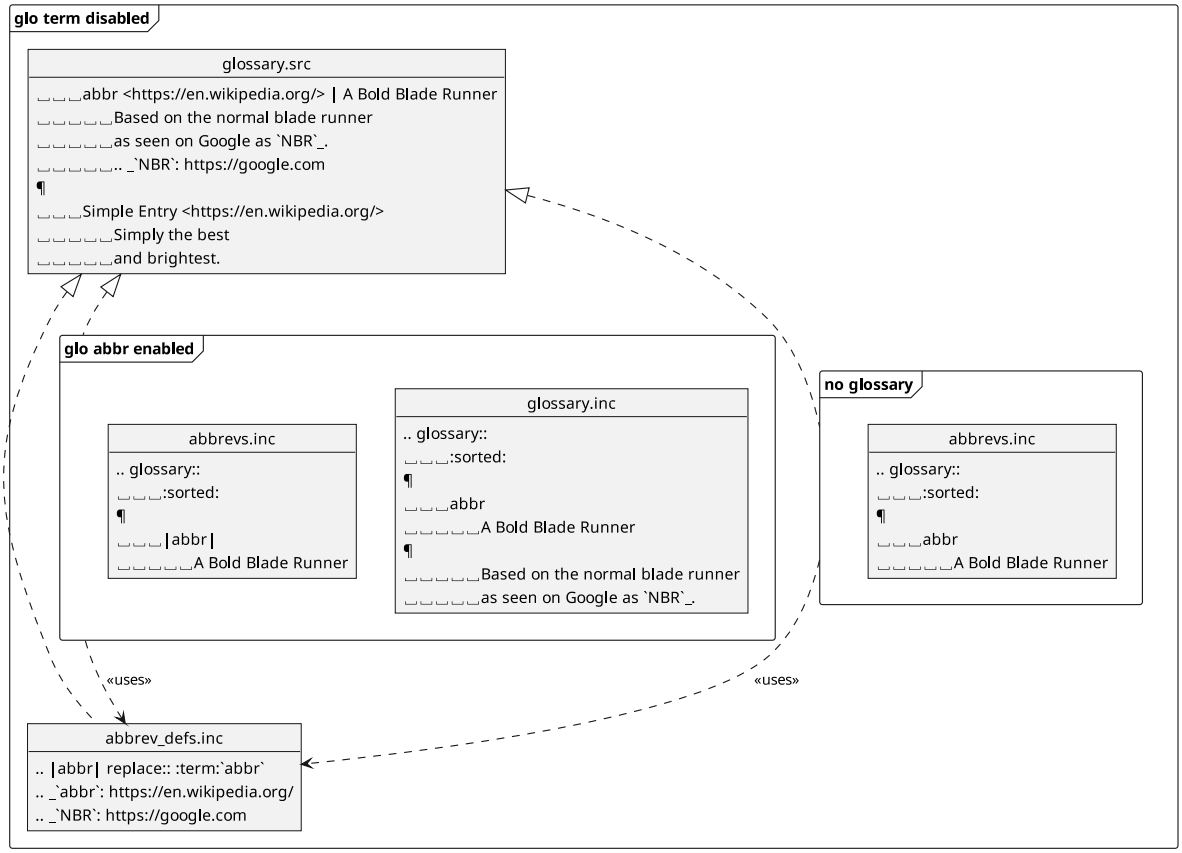


figure 6.3: Generated output files for glossary terms disabled

6.3.1 Object diagrams

6.3.2 Class diagram

6.3.3 Activity diagrams

6.3.4 State diagram

figure 6.4: Glossary class

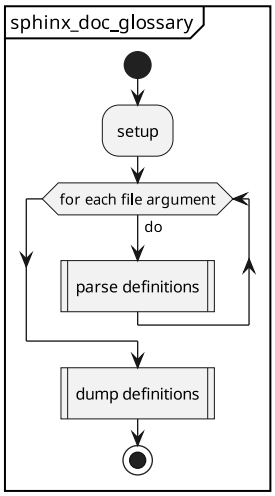
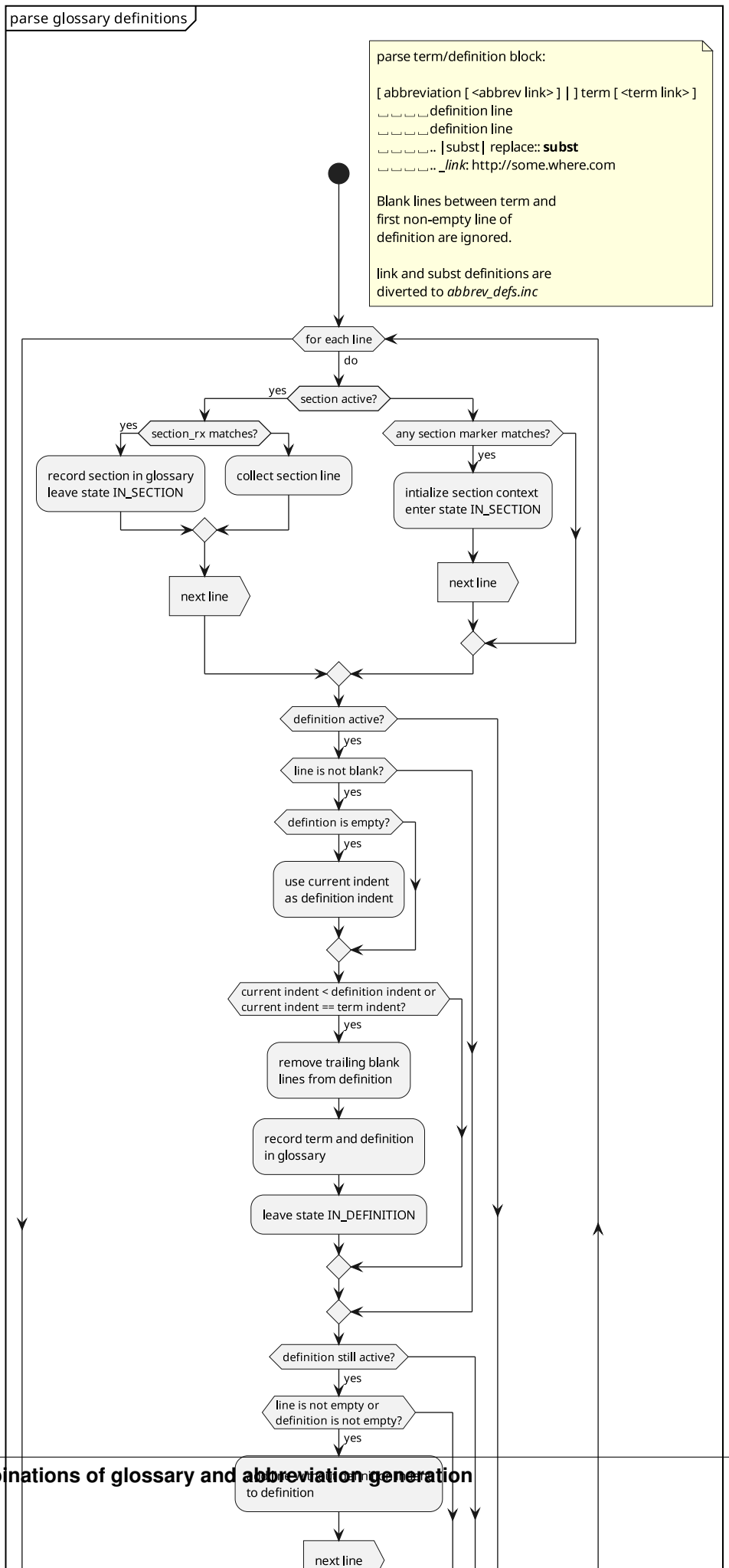


figure 6.5: Glossary parser activity diagram


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
for each line
do
yes
section active?
yes
section_rx matches?
record section in glossary
leave state IN_SECTION
collect section line
next line
any section marker matches?
yes
intialize section context
enter state IN_SECTION
next line
definition active?
yes
line is not blank?
yes
defintion is empty?
yes
use current indent
as definition indent
current indent < definition indent or
current indent == term indent?
yes
remove trailing blank
lines from definition
record term and definition
in glossary
leave state IN_DEFINITION
definition still active?
yes
line is not empty or
definition is not empty?
yes
to definition
next line

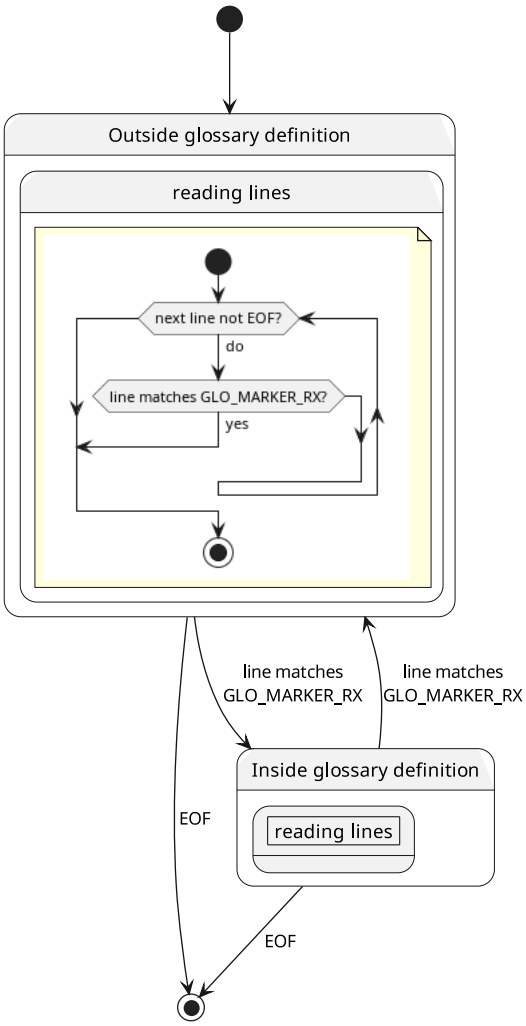


figure 6.7: Glossary parser state diagram

CHAPTER

SEVEN

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:

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

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

```
C-# C-a                 umlx-mark-activity
C-# C-b                 umlx-re-mark-buffer
C-# C-k                 umlx-unmark
C-# RET                 umlx-mark
C-# C-t                 umlx-symbol-tag-delimiter
C-# C-u                 umlx-re-mark
```

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 implicitely added by **line_diversion.py** and must not be explicitely 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*.

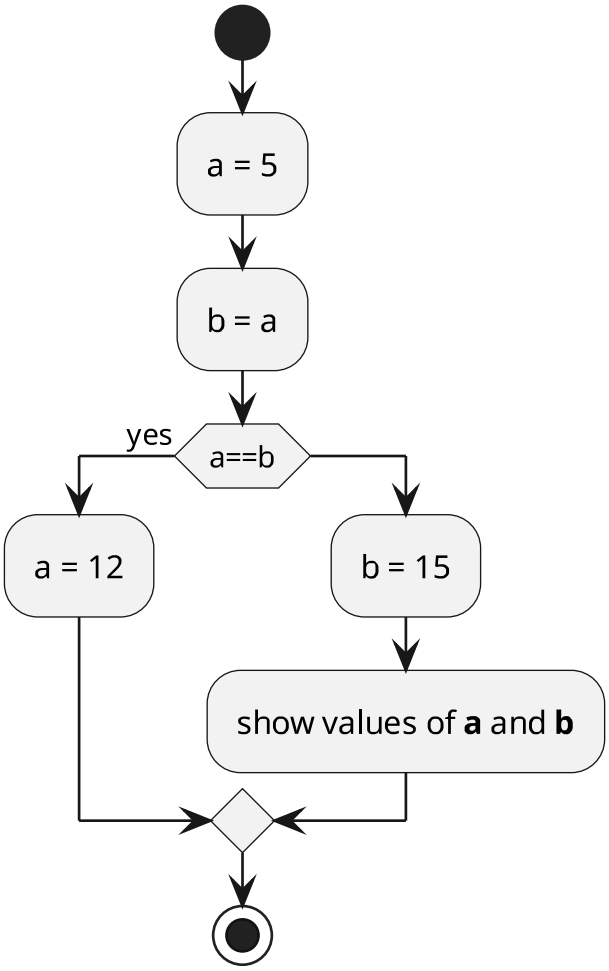
The command **umlx-extract**, which is bound to `C-# u x`, combines extraction, rendering and previewing. Without a prefix argument, the diagram for the active default marker in the current file is displayed.

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

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

With a positiv 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 extractet and rendered with `C-u 0 C-# u x a20 RET` as:

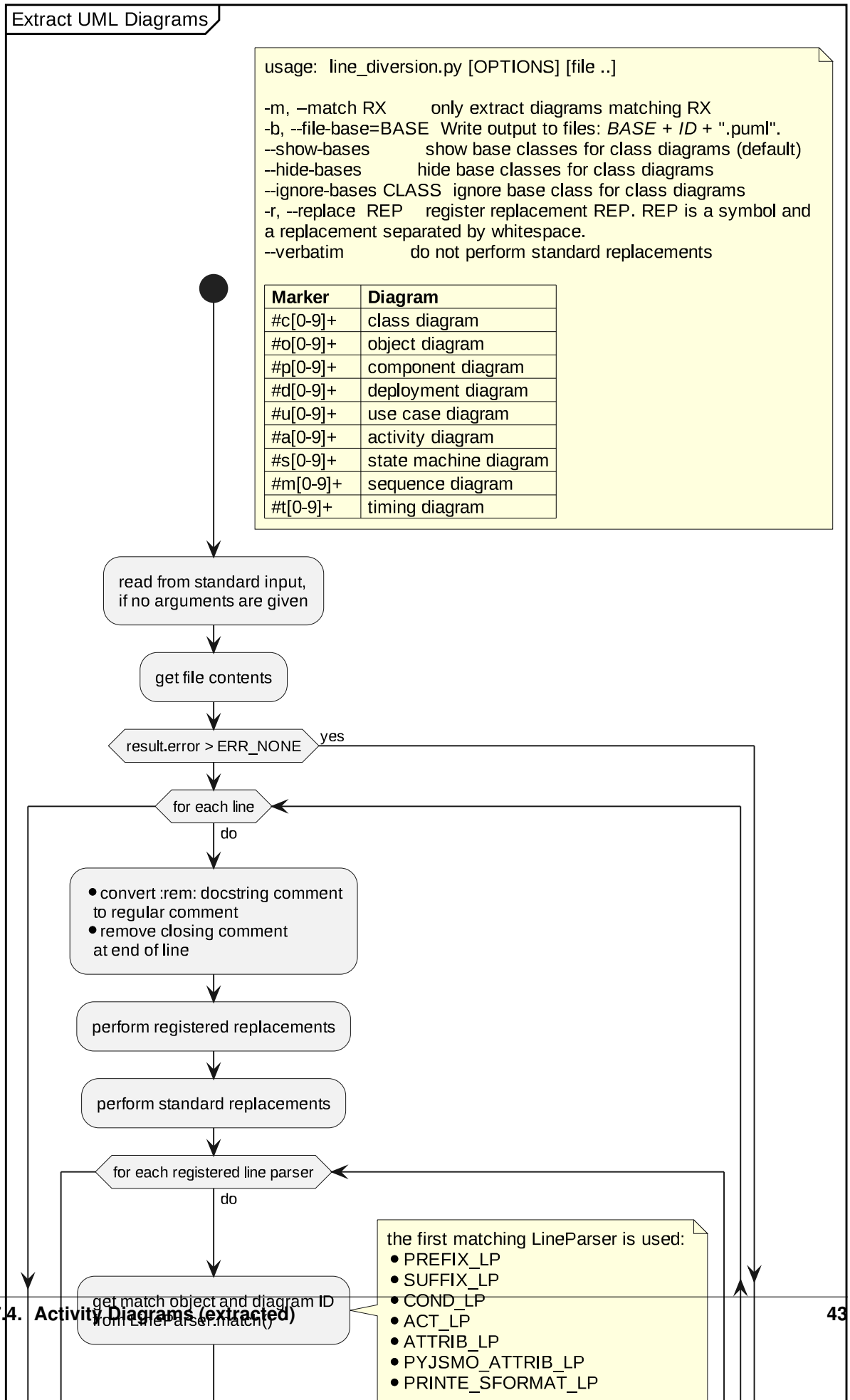


A prefix number < 0 (e.g., `C-- C-# u x` and `C-0 C-# u x`) shows the extracted PlantUML diagram definition instead of a preview. It is then very simple to walk through the output and preview diagrams with `C-c u u v`, which is bound to the command *x-plantuml-preview-current-block*. This is also convenient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

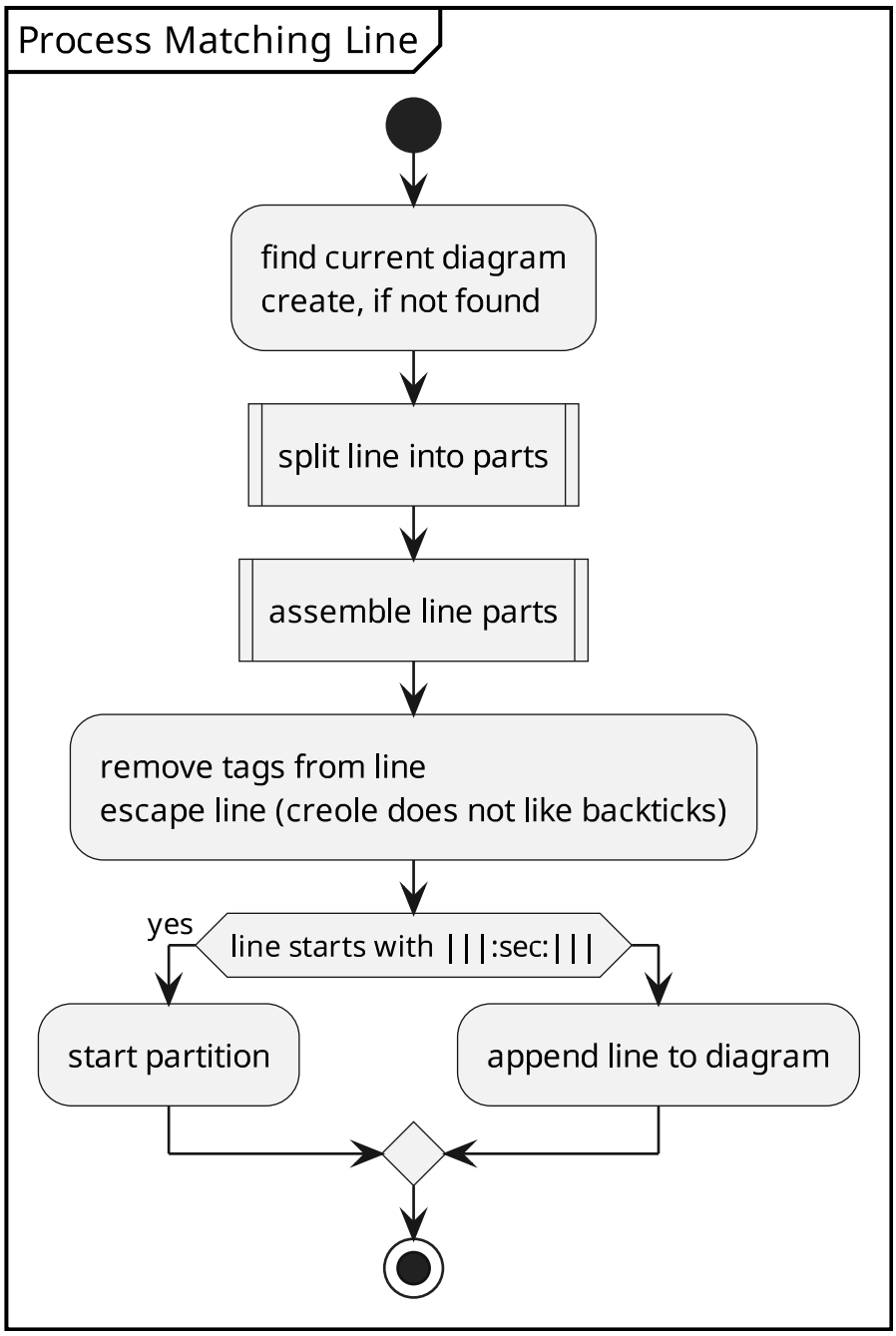
7.4 Activity Diagrams (extracted)

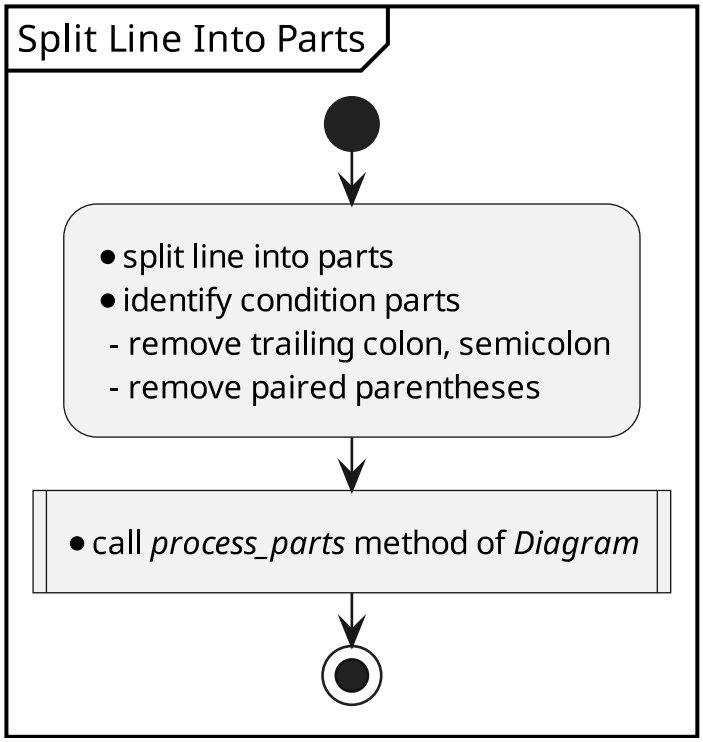
Activity diagrams for extracting UML diagrams from annotated source code:

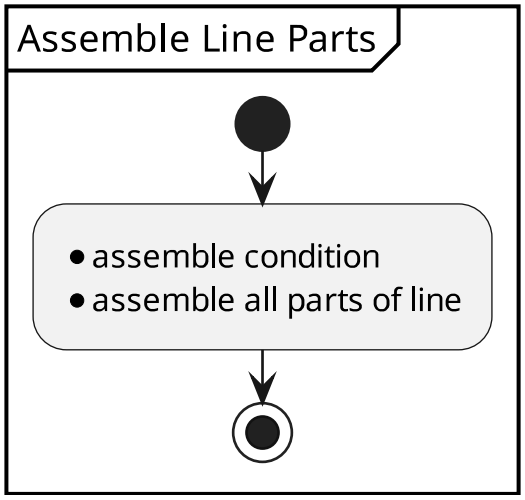
7.4.1 Extract UML diagrams



7.4.2 Process Matching Line

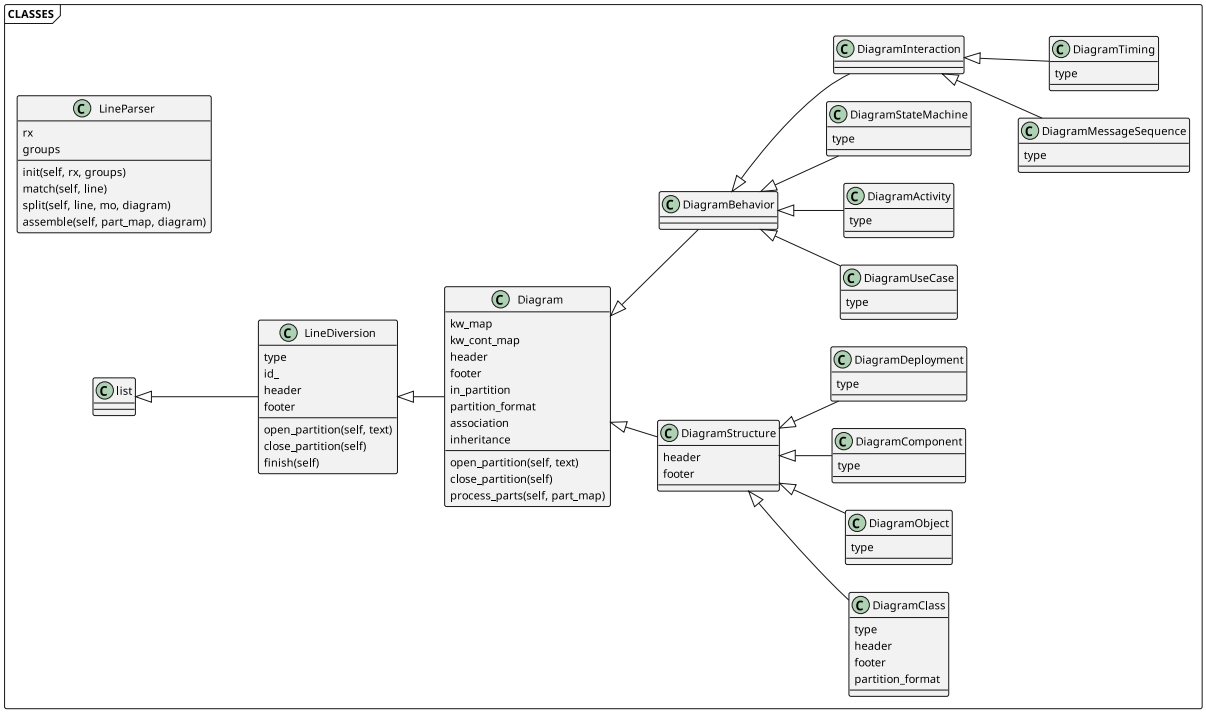






7.5 Class Diagram (extracted)

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: *BASE* + *ID* + ".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)
-r, --replace REP	register replacement REP. REP is a symbol and a replacement separated by whitespace.
–verbatim	do not perform standard replacements
-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 *python setup.py install*
–implode	implode script with adhoc
-t, –test	run doc tests

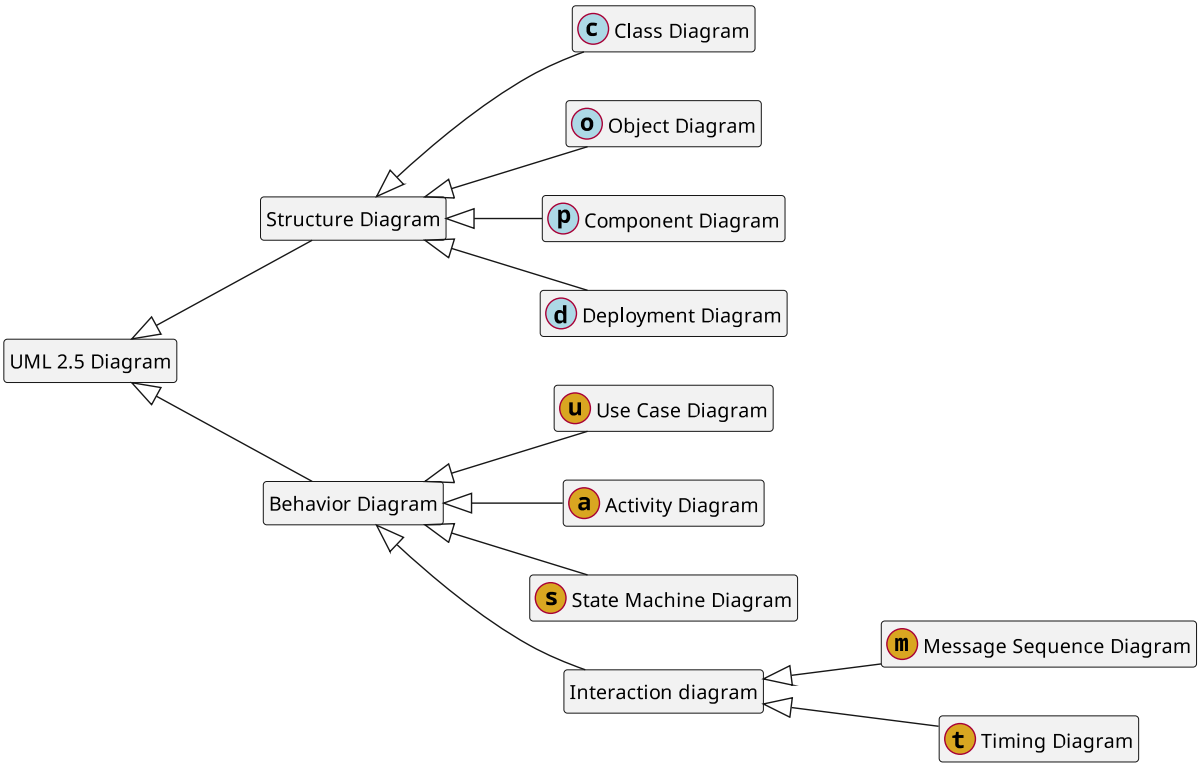
7.6.2 Module

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
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,
```

(continues on next page)

(continued from previous page)

```
        "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)
{
```

(continues on next page)

(continued from previous page)

```
    "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.

CHAPTER

EIGHT

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 deterioriation 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:

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

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.

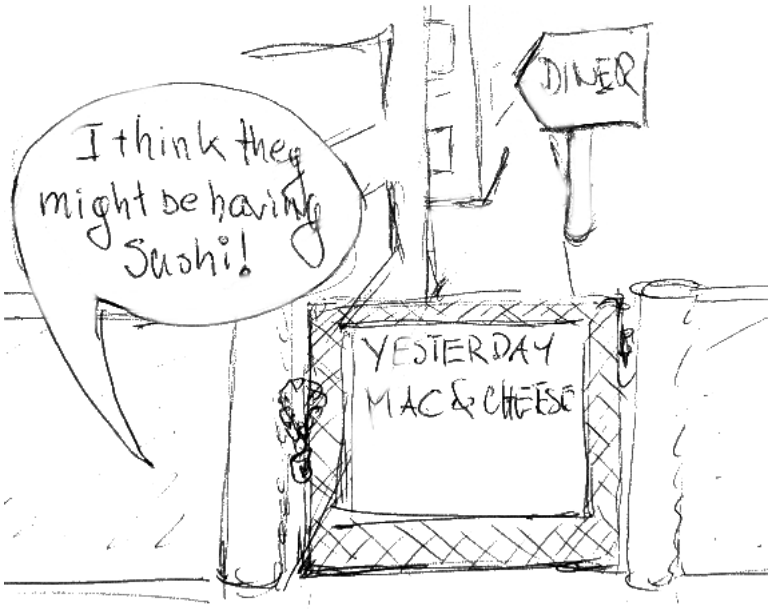


figure 8.1: I think they might be having sushi

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-referencing[1].

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.

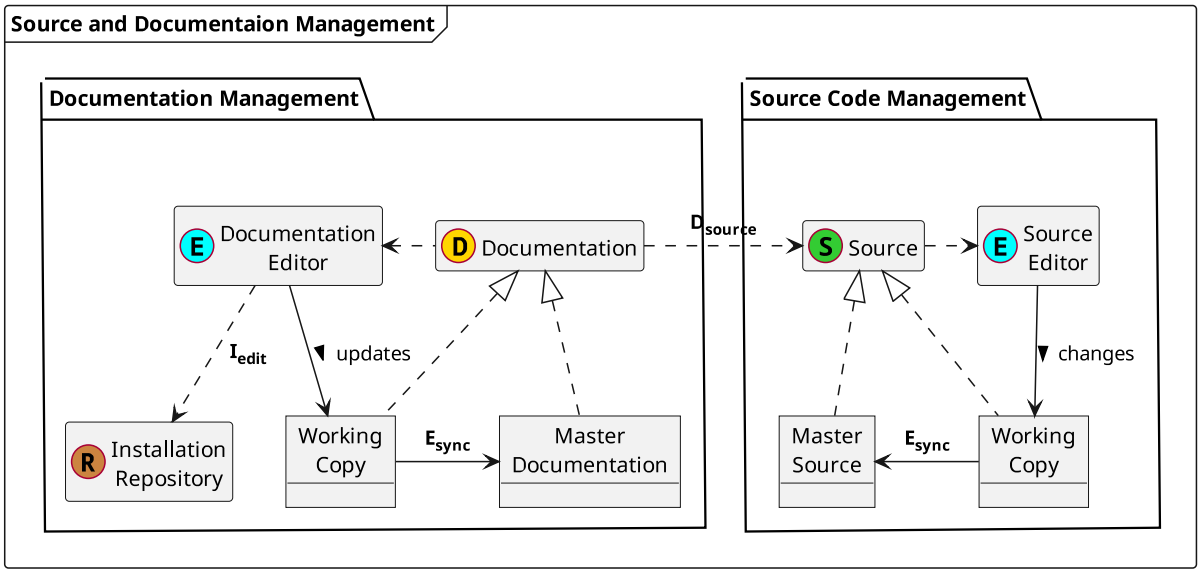


figure 8.2: Source and documentation management

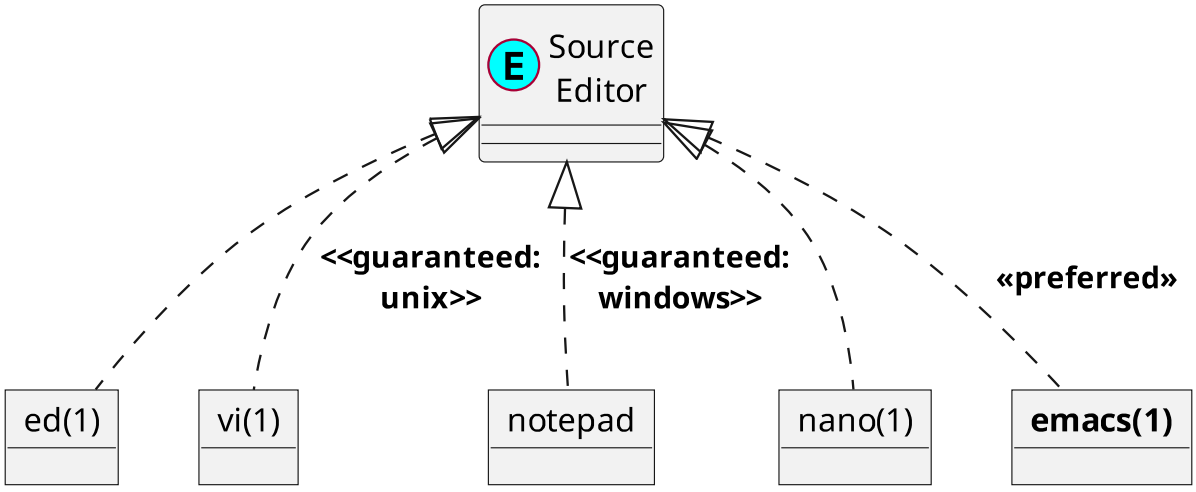


figure 8.3: Guaranteed and preferred editors

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:

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

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):

$$\left(\left(\frac{1+(E_{sync}+H_{inertia})}{2}\right)^{H_{inertia}}\right)^{-1} \tag{8.3}$$
$$L_{R_{doc}} = (1+(D_{source}+E_{sync})^{H_{inertia}})^{-1}$$

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

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

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

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

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 signifanct 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.

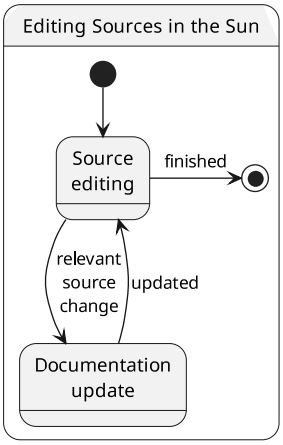


figure 8.4: Sunny day documentation update

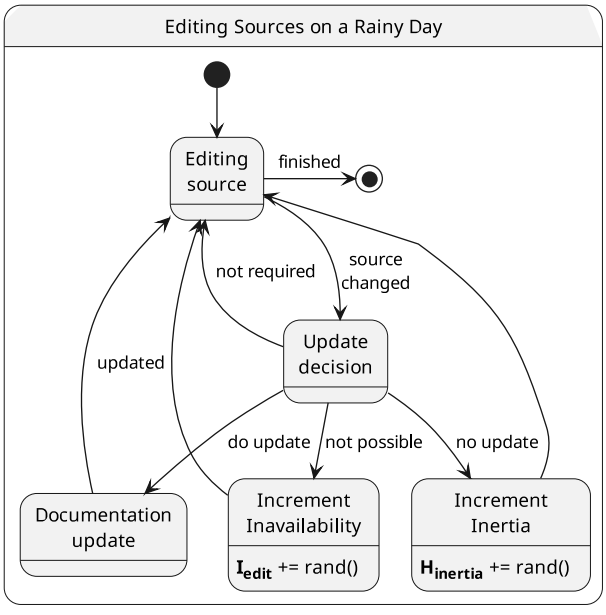


figure 8.5: Rainy day documentation update

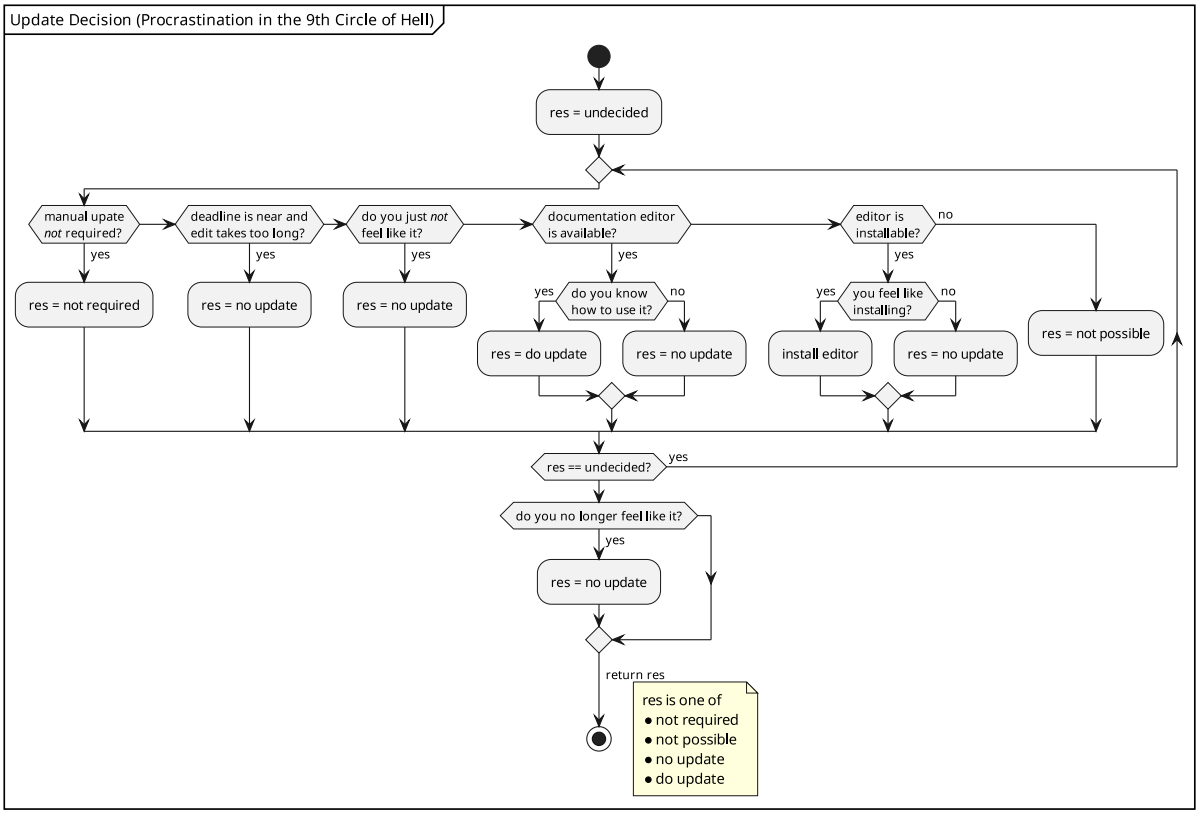


figure 8.6: Documentation update decision

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:

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

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

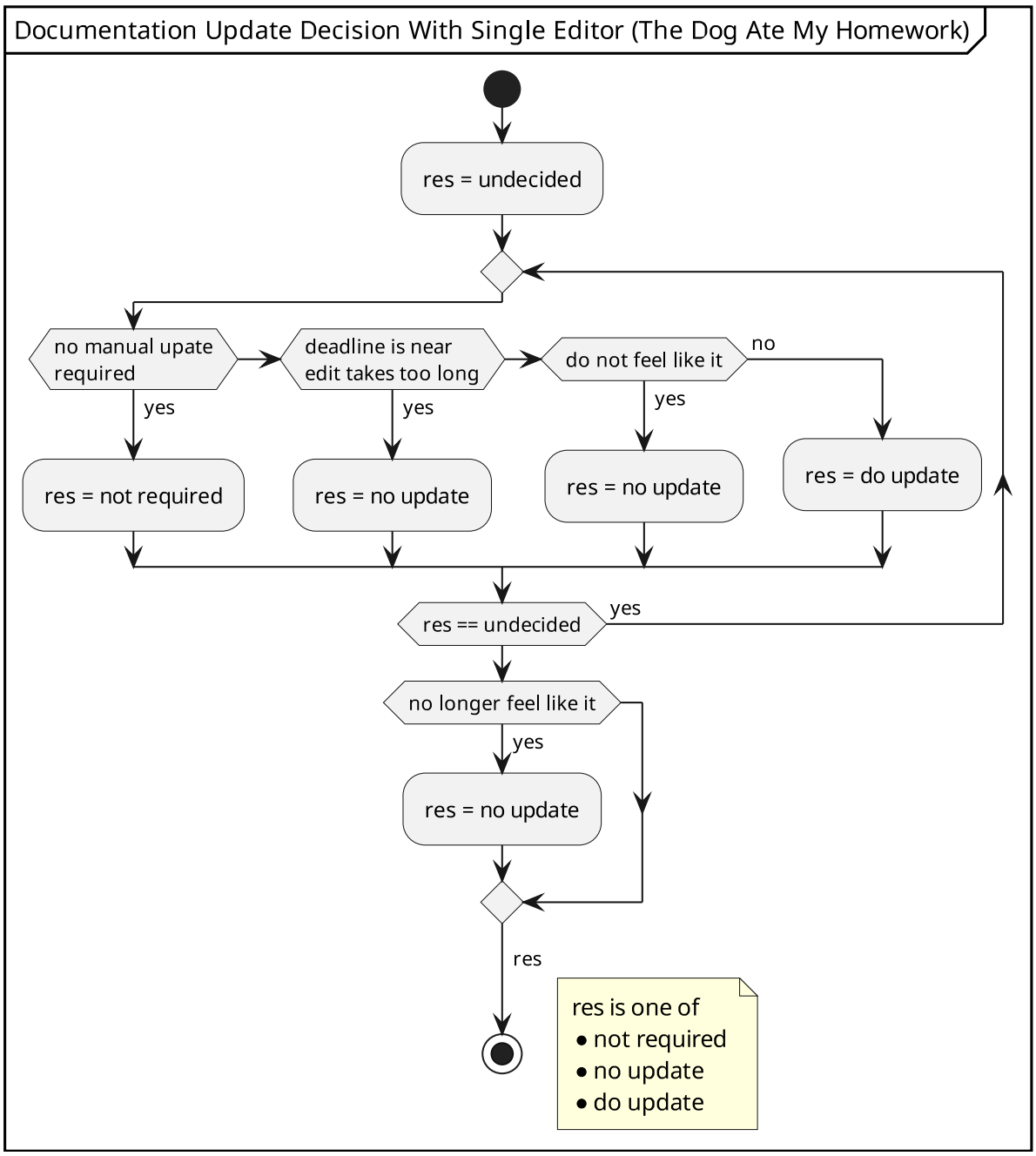


figure 8.7: Documentation Update Decision With Single Editor

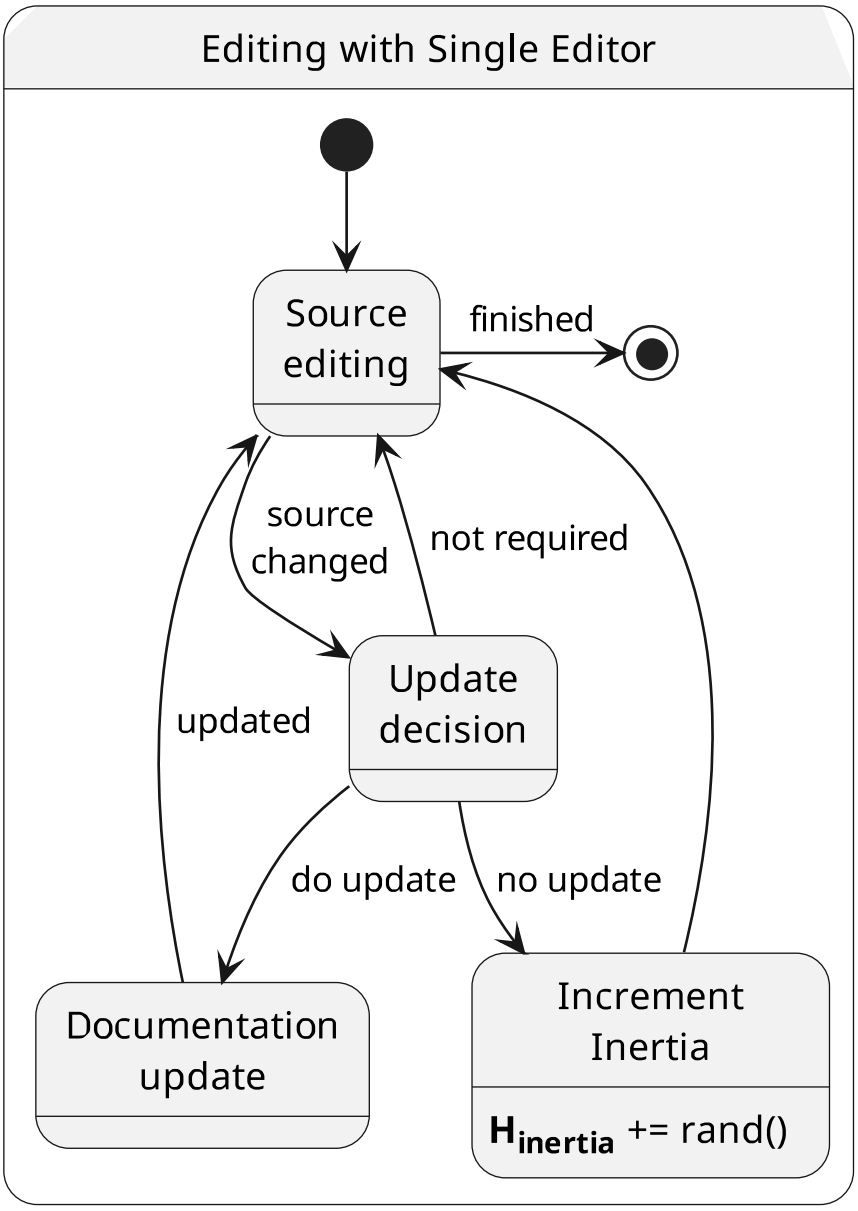


figure 8.8: Editing with single editor

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 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)

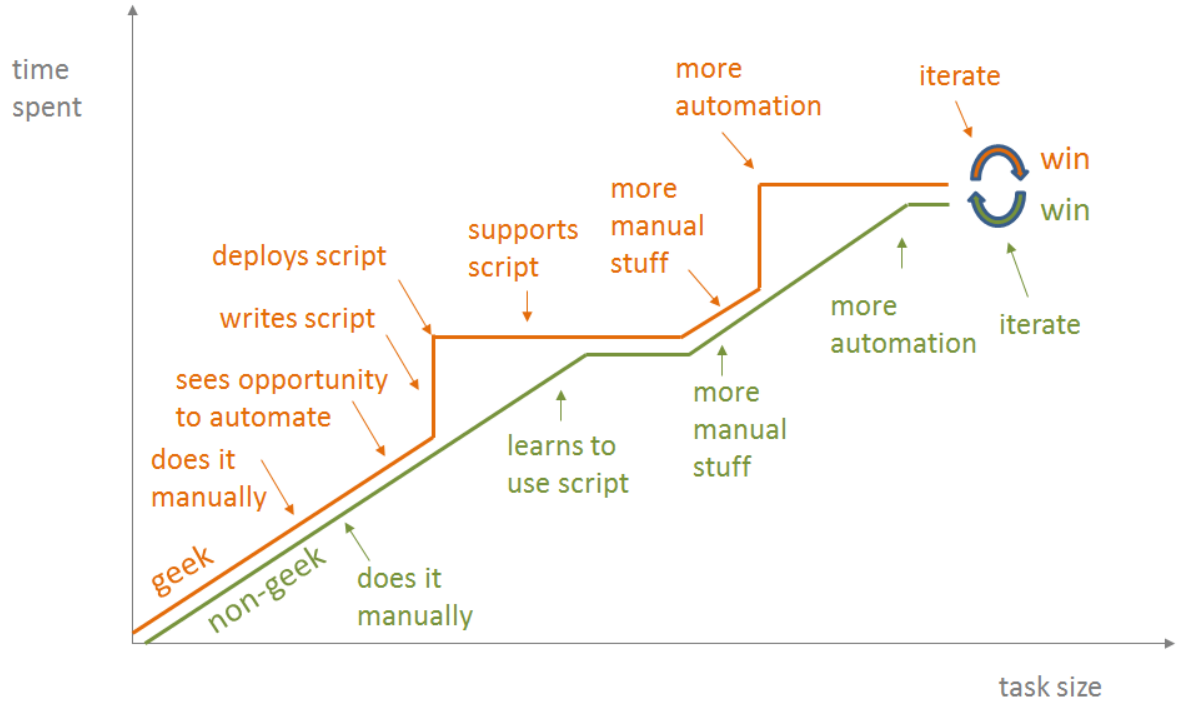
- make/scripts are memory banks for knowledge

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.



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

figure 8.9: Geeks and repetitive tasks

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 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.

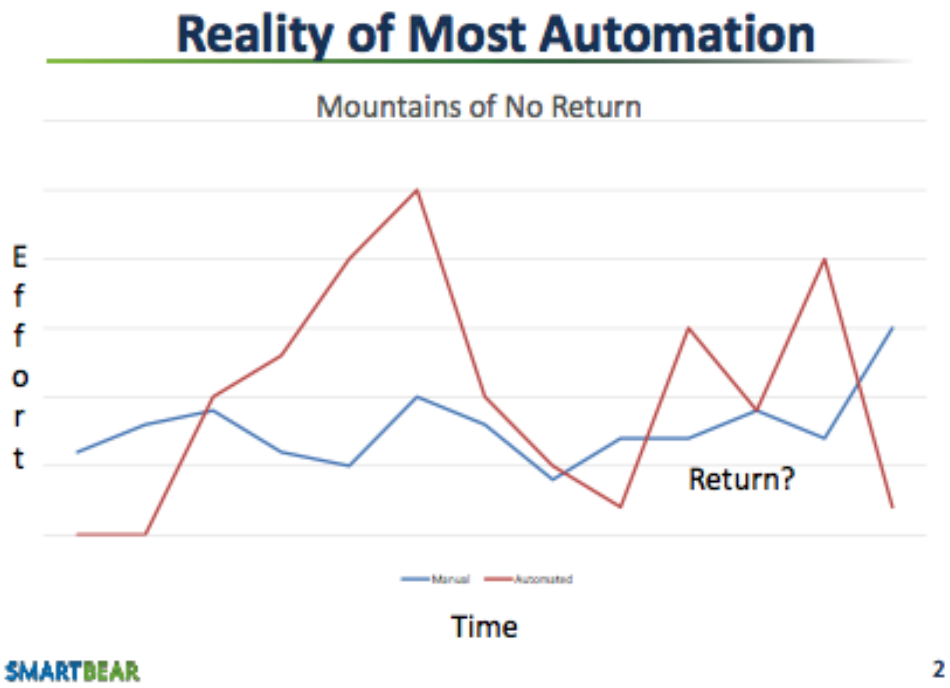
All in all Eclipse has still not reached the efficient state of the art provided by emacs(1) for decades. emacs(1) can be extended ridiculously simple with a multitude of tools glued together by Elisp.

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

CHAPTER

NINE

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 pacth `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

CHAPTER

TEN

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!

CHAPTER

ELEVEN

VCS - GIT

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*, everday 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.haegebaert.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
```

CHAPTER

TWELVE

UNIFIED MODELING LANGUAGE

12.1 UML Introduction

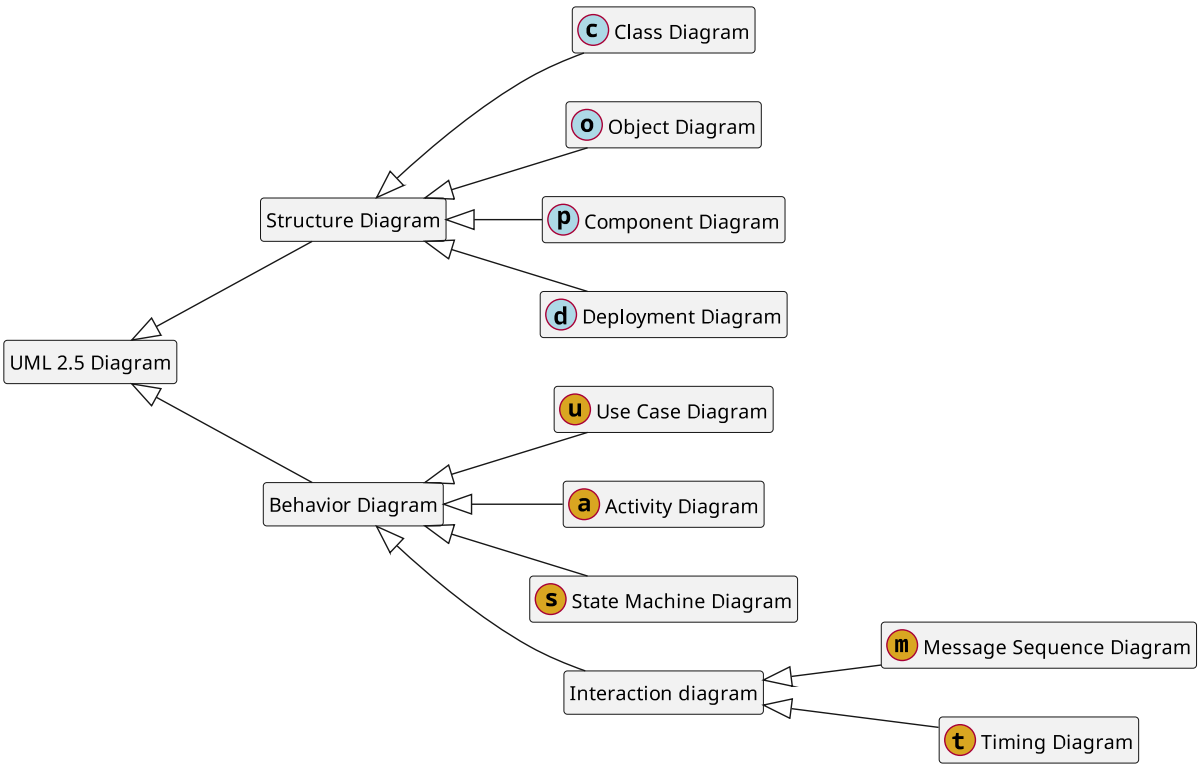
UML (Unified Modeling Language) is an ISO standardized method for describing programs. Its resulting diagrams are widely taught and should therefore be used as much as possible. The standard is mostly about the inner mechanism of the modeling language and not so much about the diagrams, however, the 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 conscise 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 defintions, 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 negative[1].

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 be 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*

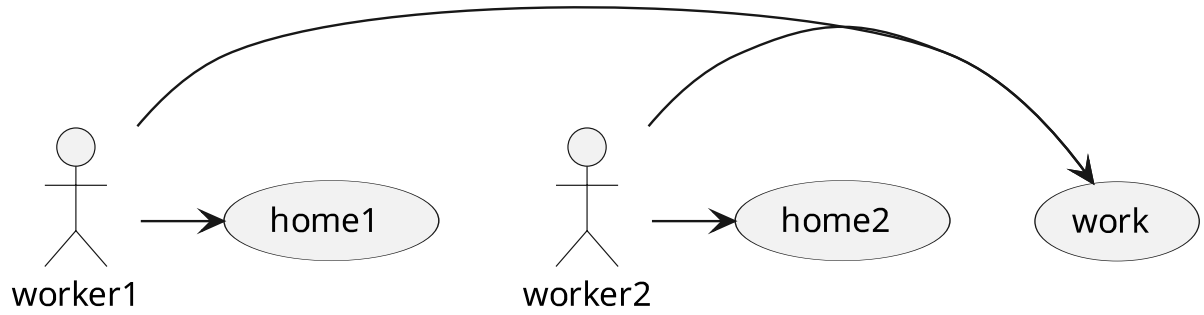
12.2.1 Use Case Diagram

> 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.

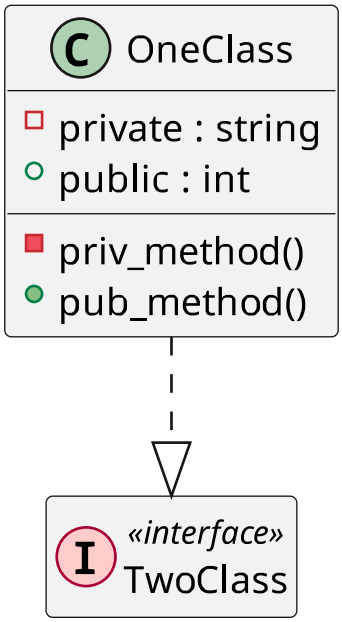
12.2.2 Class Diagram

> 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

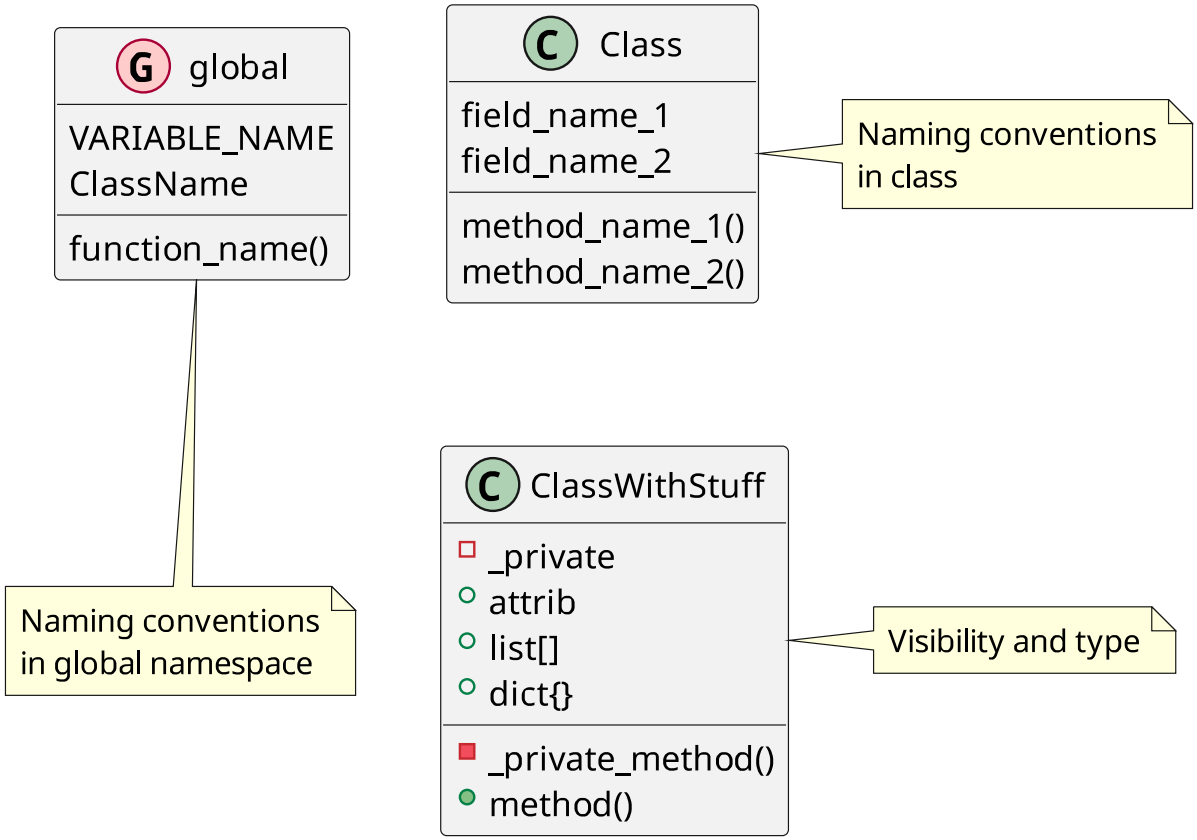
PlantUML offers various association types:

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	`+--`	

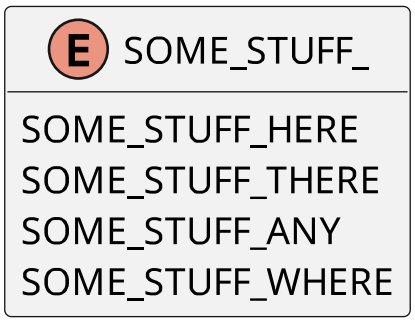
It is possible to replace `--` by `..` for broken lines.

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
```

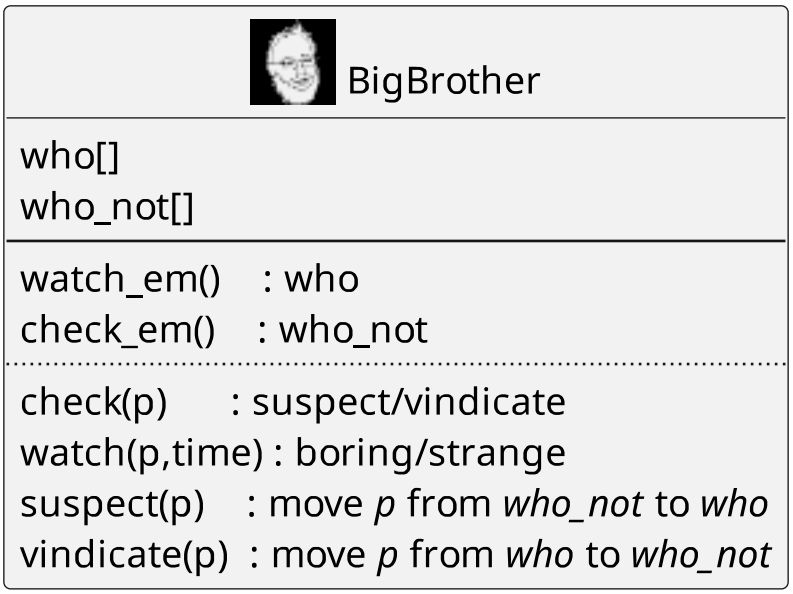
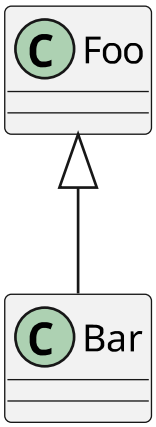
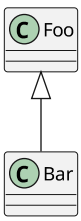


figure 12.1: Big Brother is watching you

or specify path to external PlantUML file:



You can specify height, width, scale and align:



You can specify a caption:

|:todo:| describe *uml* directive explicitely

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

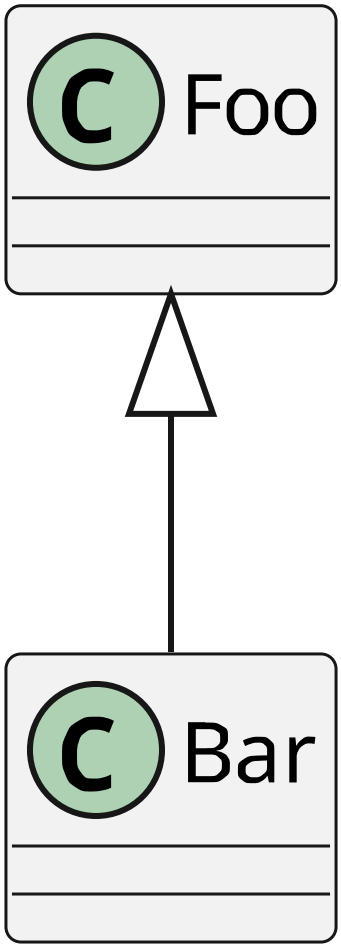


figure 12.2: Caption with **bold** and *italic*

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)

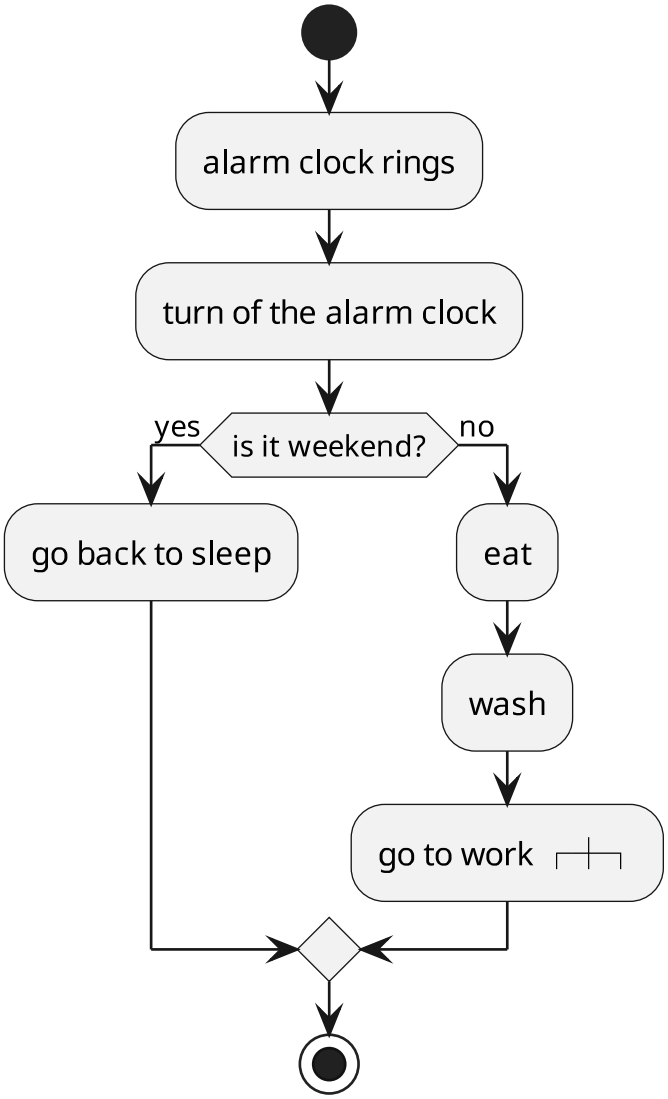
Note: It is **not** a **data flow diagram!**

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
```

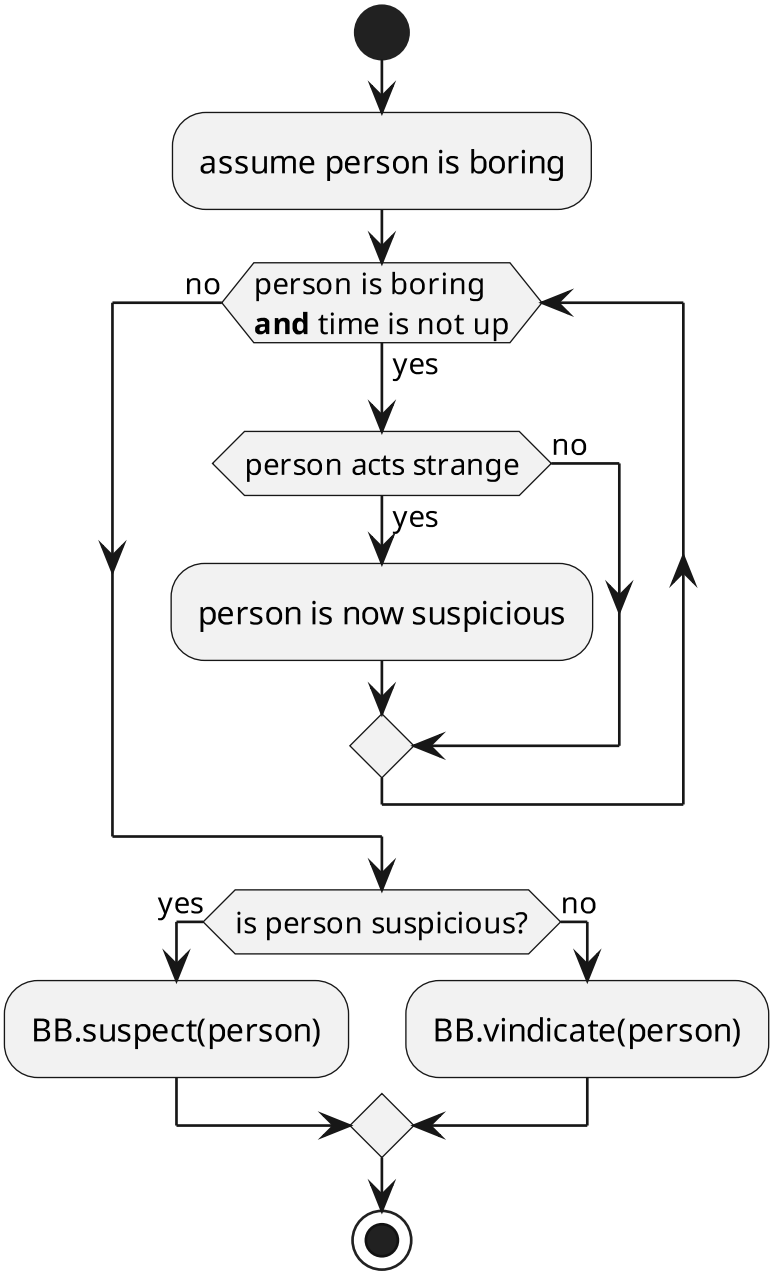
This is how it is rendered in Sphinx:



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

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

   state   create as "Create UML Diagram"
```

(continues on next page)

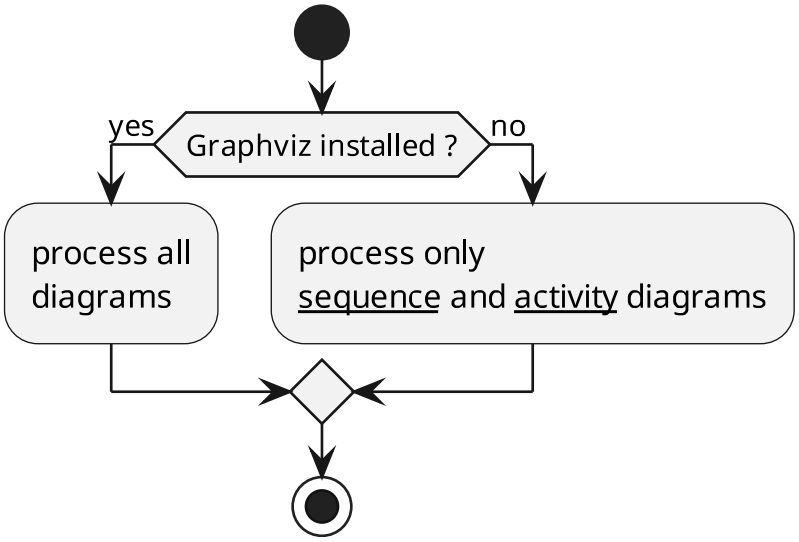
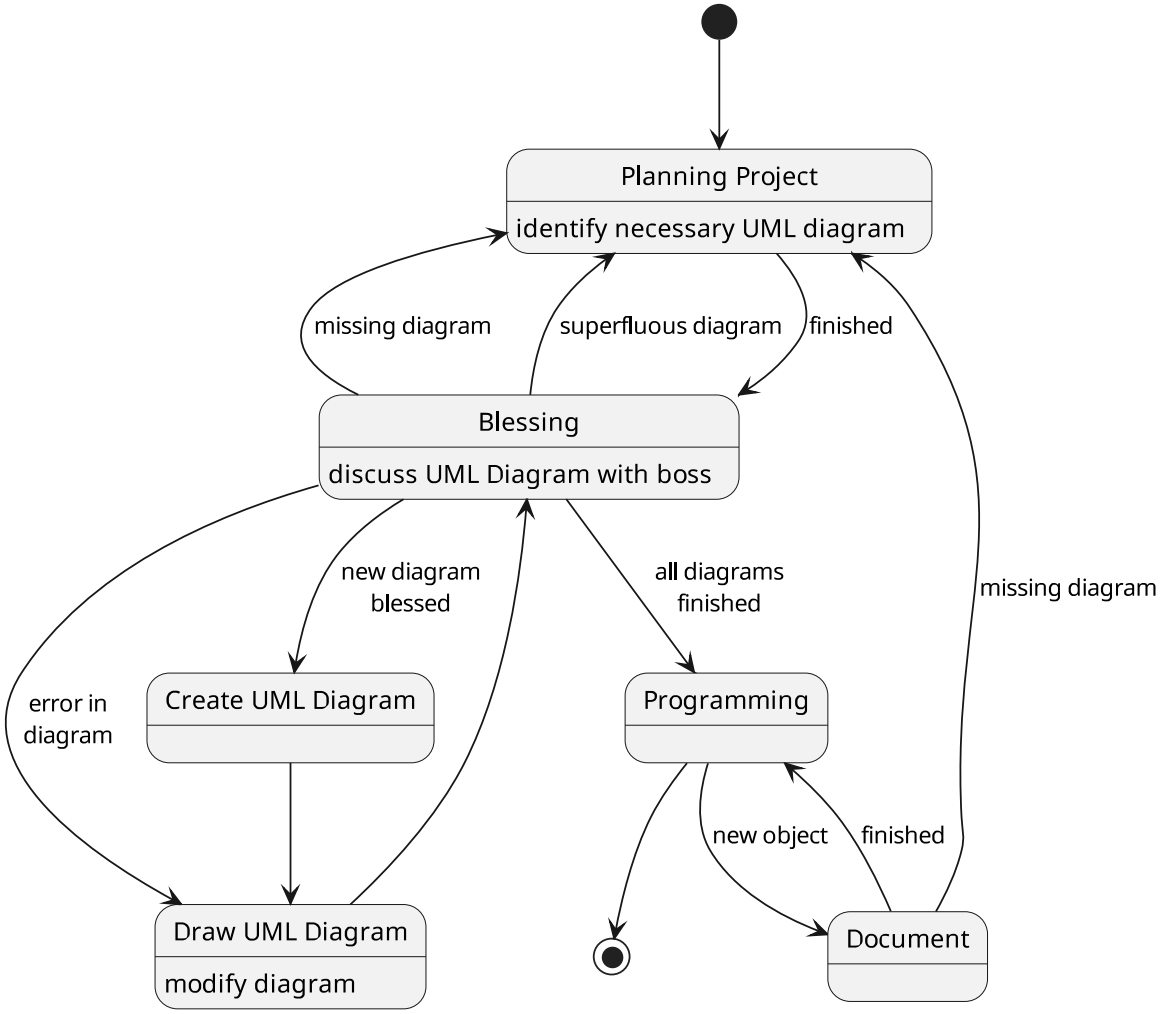


figure 12.3: Activitiy of the day

(continued from previous page)

```
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
```

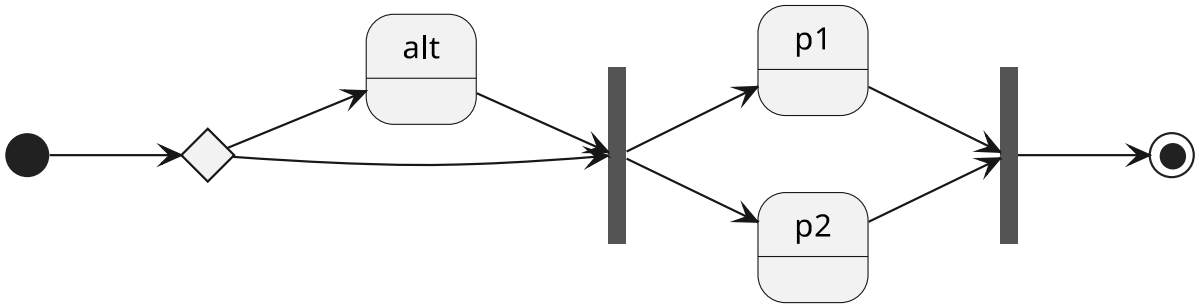
This is how it is rendered in Sphinx:



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:



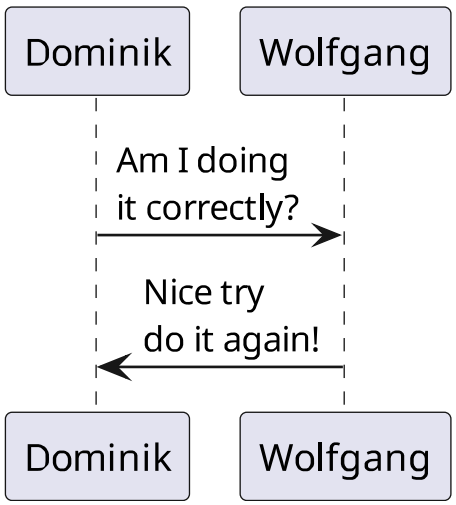
12.2.5 Sequence Diagram

> *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:

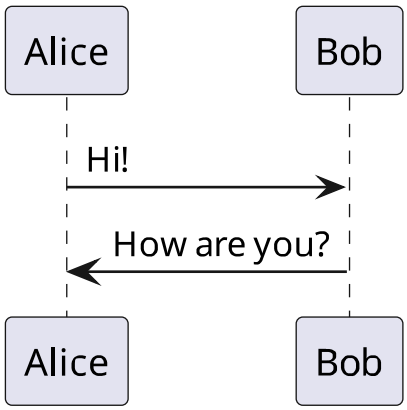
```
.. uml::

   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.

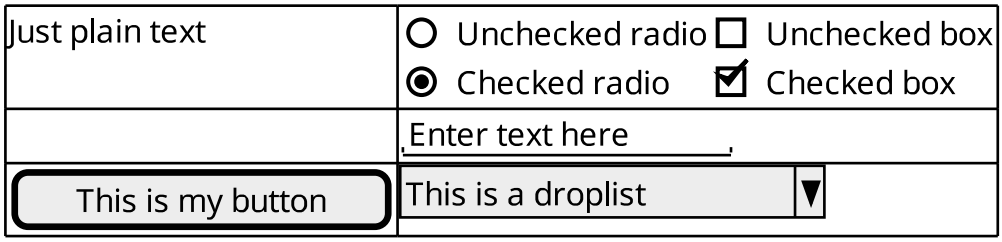
12.3.1 PlantUML Usage

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 automagically 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

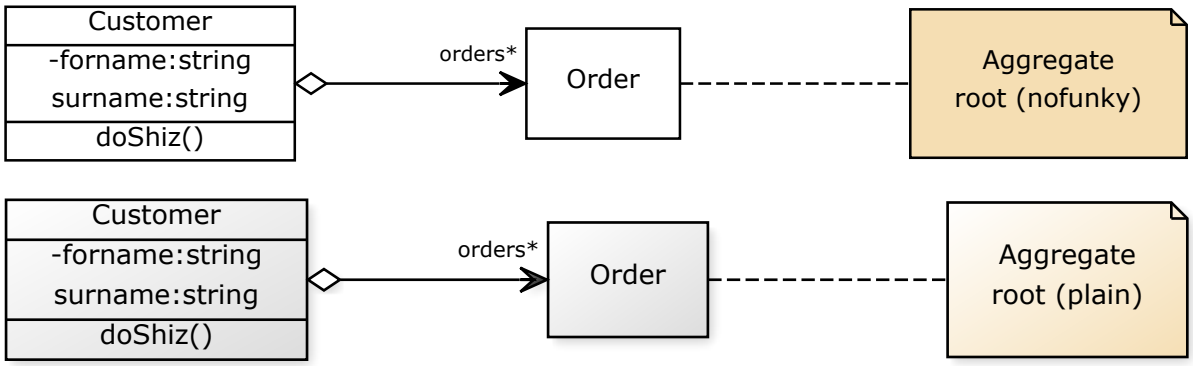
yUML is a web service only, which makes it hard to use offline.

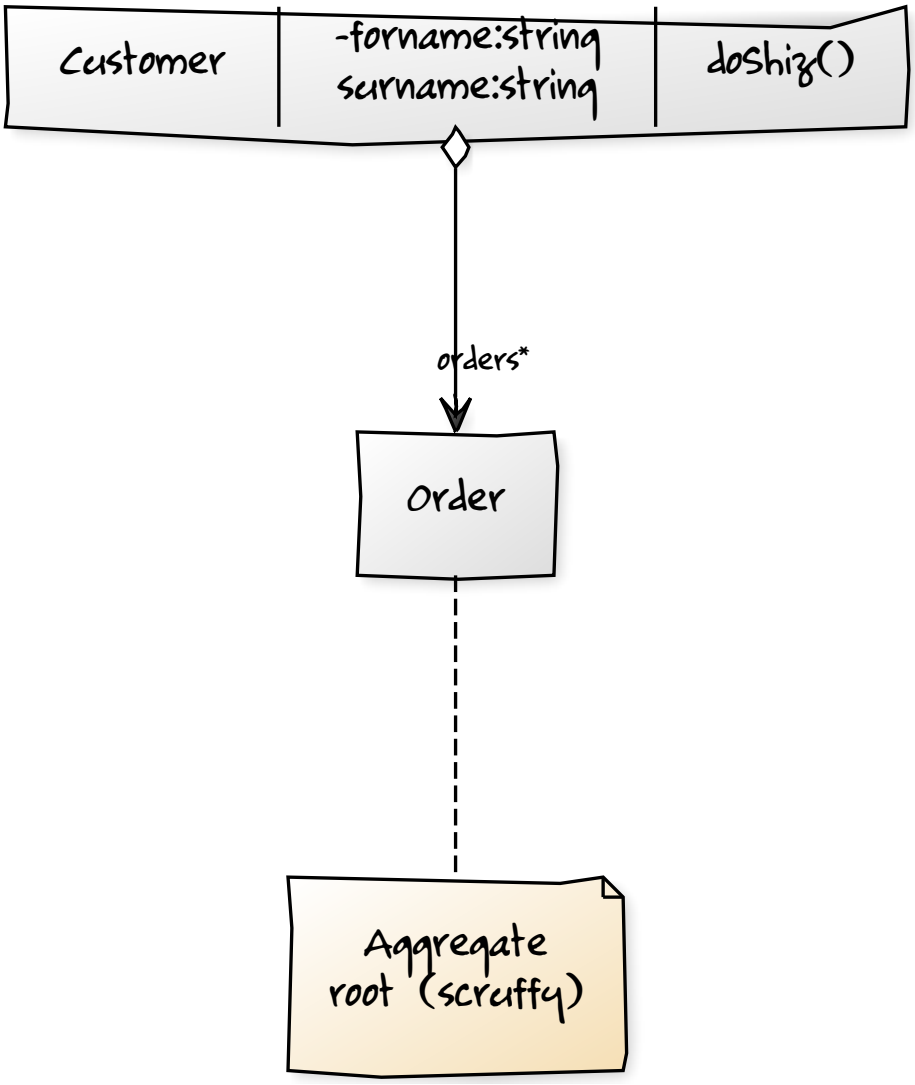
There is an unofficial command line tool for yuml and sphinx integration sphinxcontrib-yuml.

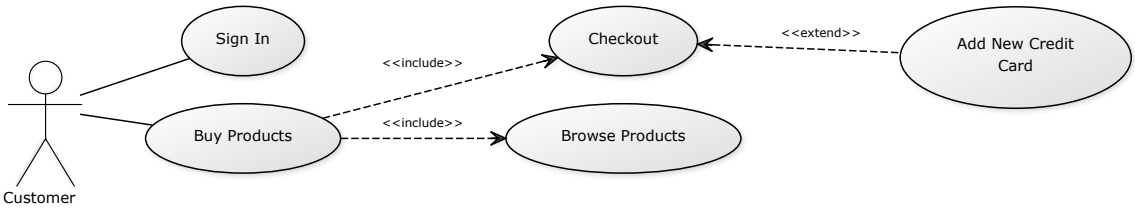
GitHub - aivarsk/scruffy is an abandoned project that does not use the web service.

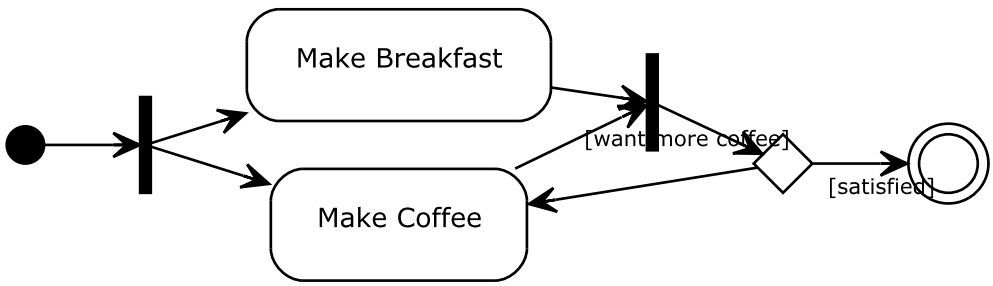
yUML provides support for class, activity and usecase diagrams:

```
.. 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}]
```




Customer
-forname:string
surname:string
doshiz()
orders*
Order
Aggregate root (scruffy)


Sign In
Checkout
<<extend>>
Add New Credit Card
<<include>>
Buy Products
<<include>>
Browse Products
Customer


Make Breakfast
[want more coffee]
[satisfied]
Make Coffee

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:

```
class_name = 'global'

func_type = type(run)
func_type_builtin = type(issequence)
module_type = type(os)

globals_ = globals()
public = globals_.get('__all__', [])
if not public:
    public = list(dkeys(globals_))

items = []
items.extend(((_sym, _val, '-') for _sym, _val in sorted(ditems(globals_)) if _sym not in
↪public))
items.extend(((_sym, globals_[_sym], '+') for _sym in public))

opt_all_syms = None

classes = []
variables = []
functions = []
for _sym, _val, _visibility in items:
    if not opt_all_syms:
        if _sym.startswith('_'):
            continue
        if _sym.startswith('__') and _sym.endswith('__'):
            continue
        if isinstance(_val, module_type):
            continue
```

(continues on next page)

(continued from previous page)

```
    if isinstance(_val, type):
        classes.append((_sym, _val, _visibility))
        continue

    if isinstance(_val, (func_type, func_type_builtin)):
        _list = functions
        _sfx = '()'
    else:
        _list = variables
        _sfx = ''
    _list.append(sformat('{0} : {1}{2}{3}', class_name, _visibility, _sym, _sfx))

printf(sformat('     class {0} << (G, #FFCCCC) >>', class_name))
for _list in variables, functions:
    printf('\n'.join(('         ' + _e for _e in _list)))
```

Or as fancy as:

```
PARENTHESES = dict((
    ('(', ')'),
    ('[', ']'),
    ('{', '}'),
    ('<', '>'),
    ('<<', '>>'),
    ('|', '|'),
    (' ', ' '),
    ('"', '"'),
    ("'", "'"),
))

__all__.append('enclose')
def enclose(elt, open=None, close=None, sep=None, forced=None):      # ||:fnc:||
    r"""enclose string in parenthesis.

    :returns: enclosed string

    :param elt:    element to be enclosed
    :param open:   opening parenthesis (default: `(`)
    :param close:  closing parenthesis (default: :data:`PARENTHESIS`.get(open, '')
    :param sep:    separator after open and before close, (default: '')
    :param forced: add parentheses, even if they already exist.

    >>> printf(enclose('abc'))
    (abc)

    >>> printf(enclose('(abc'))
    (abc)

    >>> printf(enclose('abc)'))
    (abc)

    >>> printf(enclose('(abc)'))
    (abc)

    >>> printf(enclose('(abc)', forced=True))
    ((abc))

    >>> printf(enclose('abc', '<<'))
    <<abc>>

    >>> printf(enclose(enclose('abc', ' '), '<<'))
    << abc >>

    >>> printf(enclose('abc', '-| '))
    -| abc

    """

    if open is None:
        open = '('
```

(continues on next page)

(continued from previous page)

```
    if close is None:
        close = PARENTHESES.get(open, '')
    if sep is None:
        sep = ''

    enclosed = []
    if forced or (open and not re.match('\\s*' + re.escape(open), elt)):
        enclosed.append(open)
        if open:
            enclosed.append(sep)

    enclosed.append(elt)
    if forced or (close and not re.search(re.escape(close) + '\\s*$', elt)):
        if close:
            enclosed.append(sep)
        enclosed.append(close)
    return ''.join(enclosed)

__all__.append('join_lists_flatten')
def join_lists_flatten(lists, sep):                         # ||:fnc:||
    r"""Insert separator between elements of a list.

    :returns: list where each group of elements is separated from the next
      group of element with `sep`.

    :param lists: a list of element groups (lists). All elements of
      group are appended as single elements to the result list.
    :param sep: separater inserted betwwen two groups of elements.

    """
    result = []
    if lists:
        result.extend(lists[0])
    for _list in lists[1:]:
        result.append(sep)
        result.extend(_list)
    return result

__all__.append('plantuml_format_class')
def plantuml_format_class(name, sections=None, circle=None, stereotype=None, alias=None,␣
↪sep=None): # ||:fnc:||
    r"""
    :returns: class formatted for a `PlantUML`_ class diagram.

    :param name:       class name
    :param sections:   list of class sections, each section is a list of attribute/operation␣
↪lines.
    :param circle:
    :param stereotype:
    :param alias:      class alias

    .. _`PlantUML`: http://plantuml.com
    """

    if sections is None:
        sections = []

    formatted = []

    _class_param = []
    if circle and circle.strip():
        _class_param.append(enclose(circle, '('))
    if stereotype and stereotype.strip():
        _class_param.append(stereotype)
    class_param = ' '.join(_class_param)
    if class_param:
        class_param = enclose(class_param, '<<', sep=' ')

    if alias:
        name = enclose(name, '"')
        class_alias = 'as ' + alias
```

(continues on next page)

(continued from previous page)

```
    else:
        class_alias = ''

    if sep is None:
        sep = ''

    formatted.append(sformat('class {0} {{', ' '.join((_s for _s in (name, class_alias, class_
↪param) if _s))))

    _first = True
    for _list in sections:
        if _list:
            if _first:
                _first = False
            else:
                formatted.append('  ' + sep)
            formatted.extend(('  ' + _e for _e in _list))

    formatted.append('}')
    return '\n'.join(formatted)

class_name = 'global'

func_type = type(run)
func_type_builtin = type(issequence)
module_type = type(os)

globals_ = globals()
public = globals_.get('__all__', [])
if not public:
    public = list(dkeys(globals_))

items = []
items.extend(((_sym, _val, '-') for _sym, _val in sorted(ditems(globals_)) if _sym not in␣
↪public))
items.extend(((_sym, globals_[_sym], '+') for _sym in public))

opt_all_syms = None

import pyjsmo
classes = []
variables = pyjsmo.OrderedDict((
    ('-', []),
    ('+', []),
))
functions = pyjsmo.OrderedDict((
    ('-', []),
    ('+', []),
))
for _sym, _val, _visibility in items:

    if not opt_all_syms:
        if _sym.startswith('_'):
            continue
        if _sym.startswith('__') and _sym.endswith('__'):
            continue
        if isinstance(_val, module_type):
            continue

    if isinstance(_val, type):
        classes.append((_sym, _val, _visibility))
        continue

    if isinstance(_val, (func_type, func_type_builtin)):
        _list = functions[_visibility]
        _sfx = '()'
    else:
        _list = variables[_visibility]
        _sfx = ''
    _list.append(sformat('{0}{1}{2}', _visibility, _sym, _sfx))

```

(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

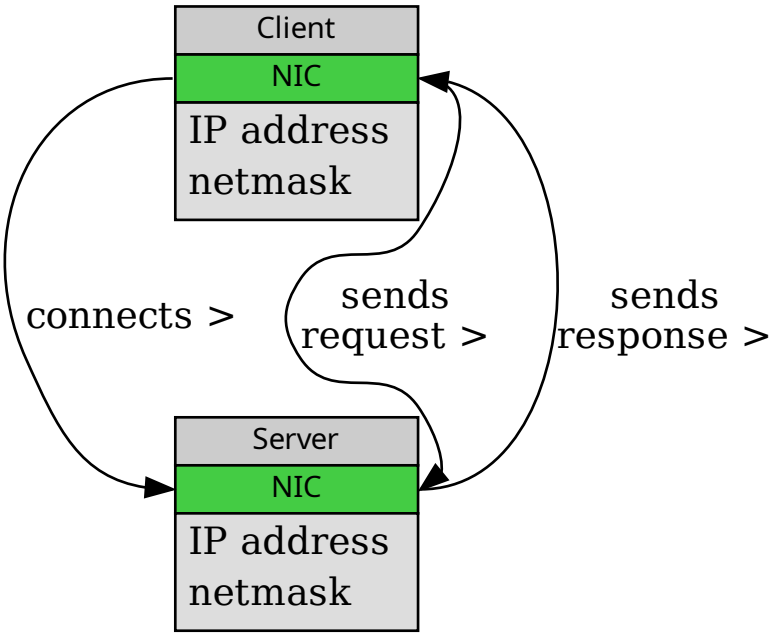
- UML is the abbreviation of unified modeling language. It is a standarizied method for descriping programms.
- The UML diagrams are easy to understand even for noprogrammers
- There are existing some UML GUI-TOOLS. There are good for noprogrammers to modeling something, but programmers should'nt use them, because the not-diagram-output is'nt 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 happends in the programm via UML, without the GUI-TOOL, the output is worthless)
- Programmers should use PlantUML to create UML diagrams.
  - PlantUML is a 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 programm, because it is quite easy to understand the associations of the programm 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 commend, 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 it's 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 relationsship, 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.


Client
NIC
IP address
netmask
connects >
sends
request >
sends
response >
Server
NIC
IP address
netmask

CHAPTER

THIRTEEN

TOOLS

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

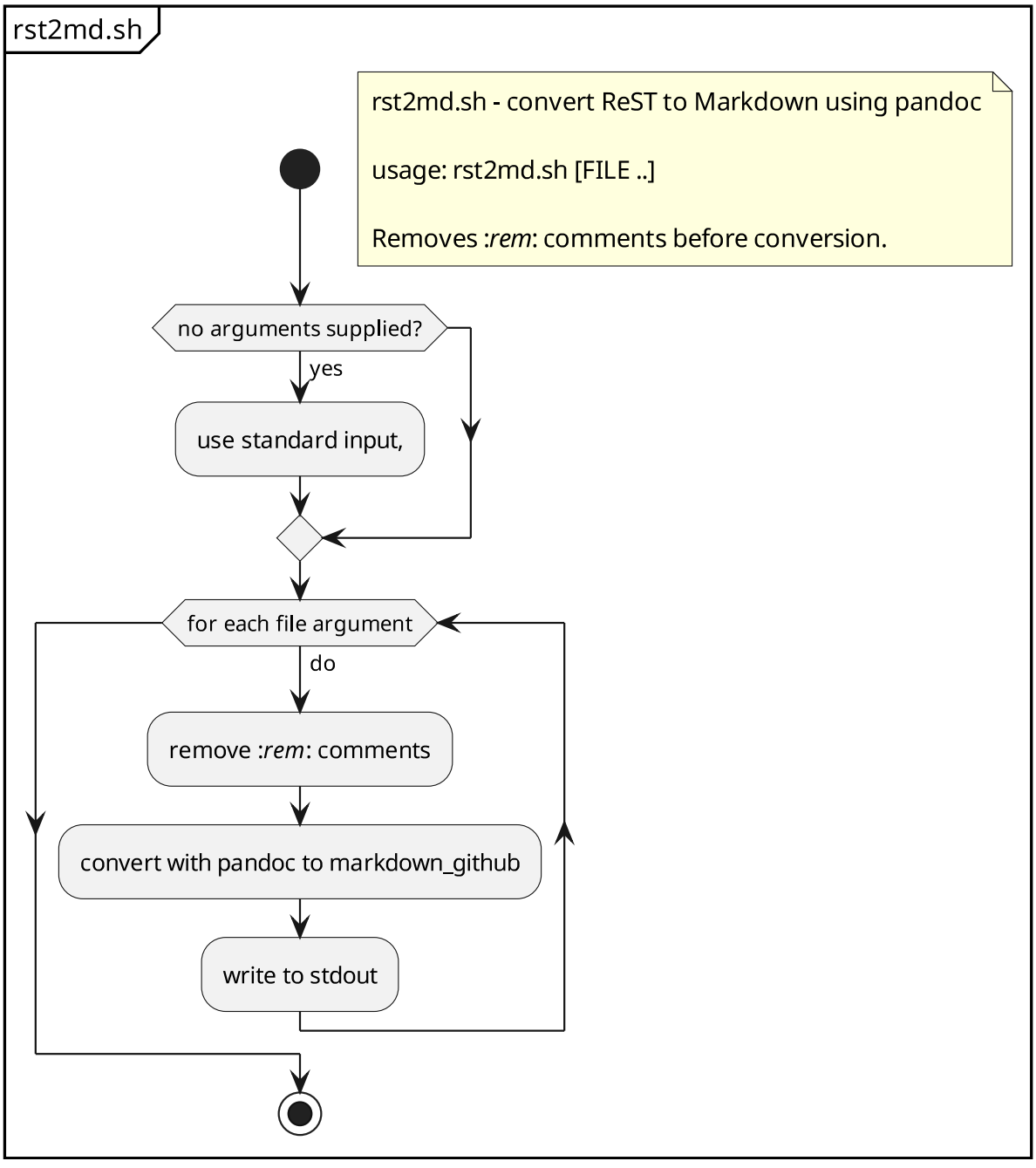
     ```
     <delegate decode="svg:decode" stealth="True" command="&quot;inkscape&quot; &quot;%s&
     ↪quot; --export-eps=&quot;%s&quot; --export-dpi=&quot;%s&quot; --export-
     ↪background=&quot;%s&quot; --export-background-opacity=&quot;%s&quot; > &quot;
     ↪%s&quot; 2>&amp;1"/>
     ```

     to

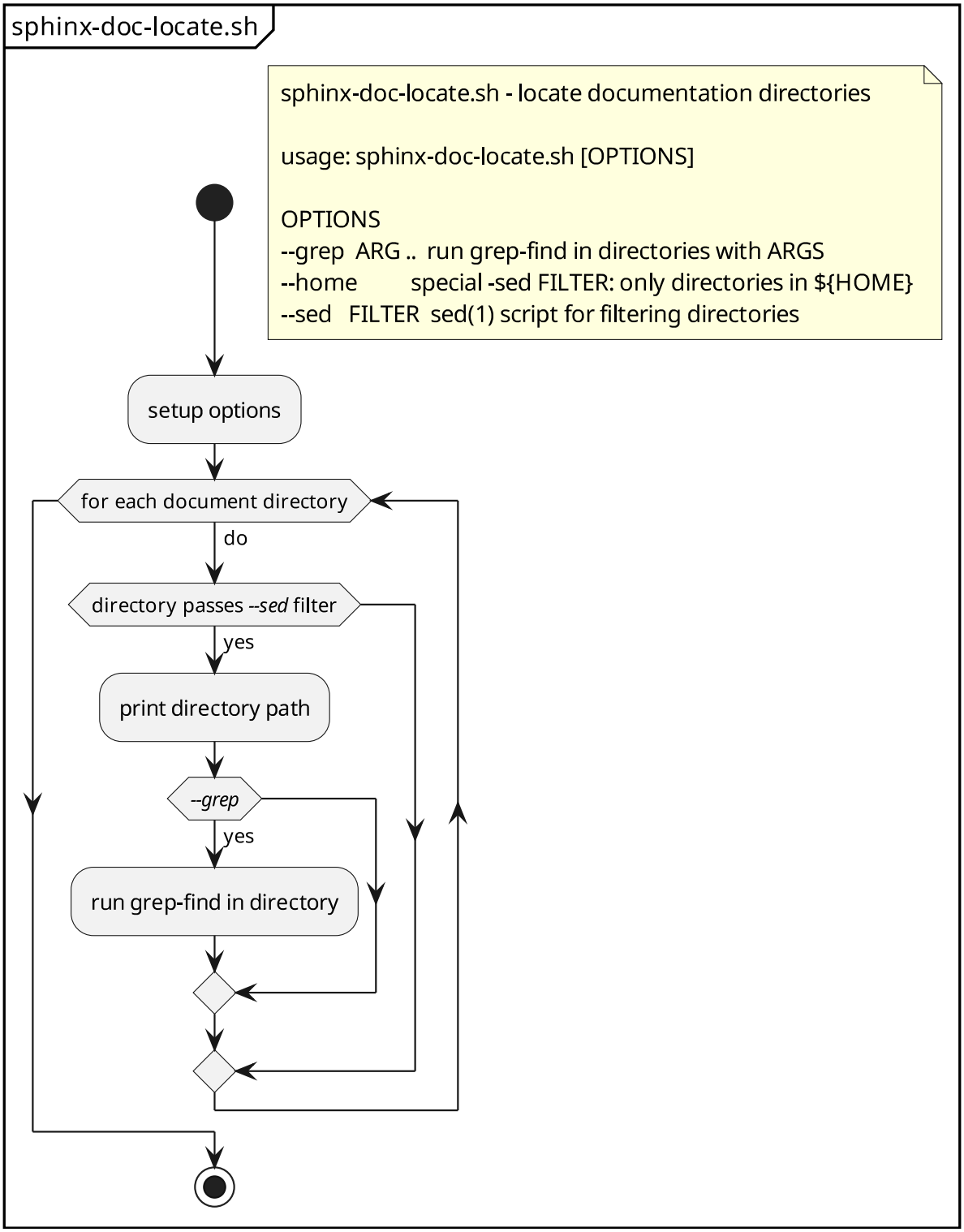
     ```
     <delegate decode="svg:decode" stealth="True" command="&quot;inkscape&quot; &quot;%s&
     ↪quot; --export-png=&quot;%s&quot; --export-dpi=&quot;%s&quot; --export-
     ↪background=&quot;%s&quot; --export-background-opacity=&quot;%s&quot; > &quot;
     ↪%s&quot; 2>&amp;1"/>
     ```

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

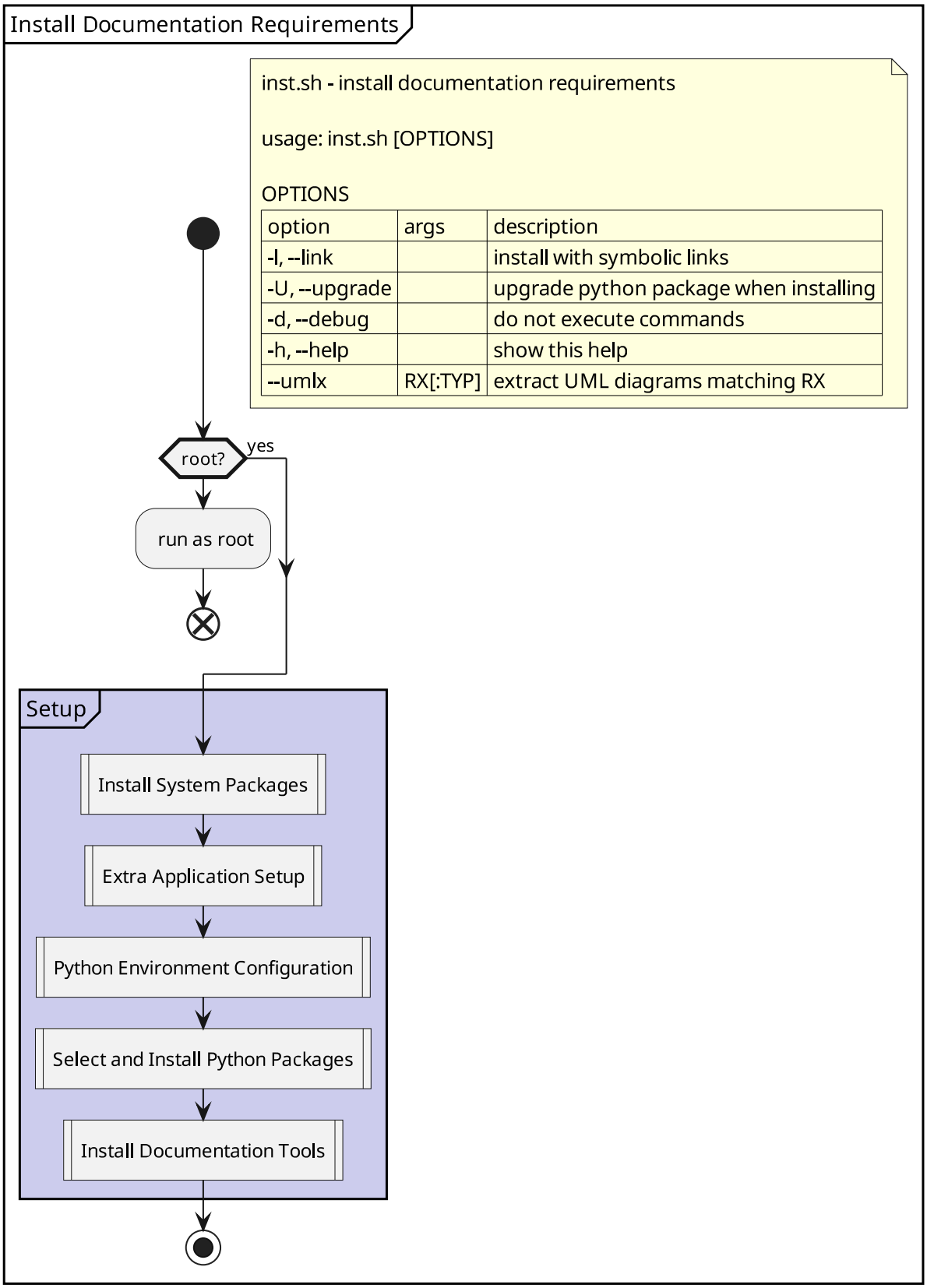
13.2 Activity Diagrams for `rst2md.sh`

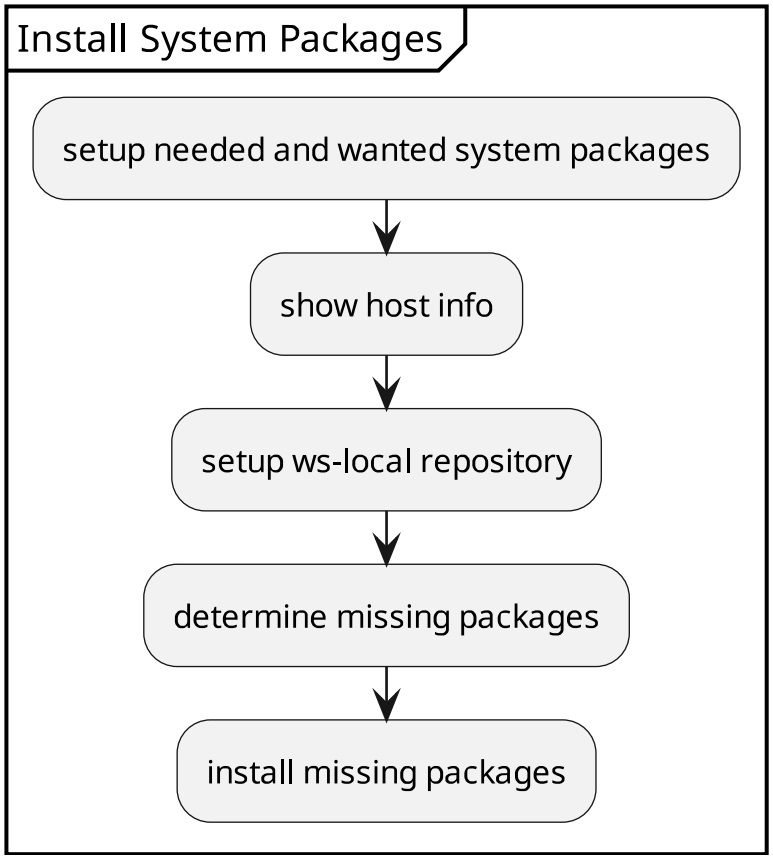


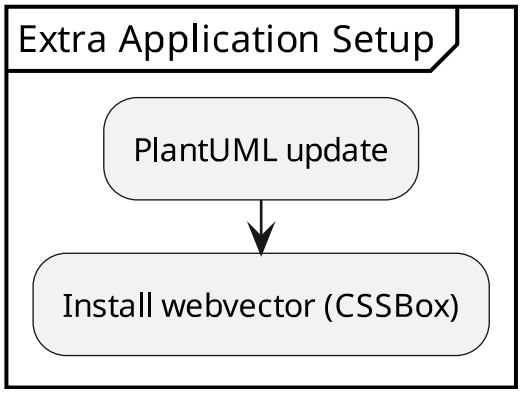
13.3 Activity Diagrams for `sphinx-doc-locate.sh`

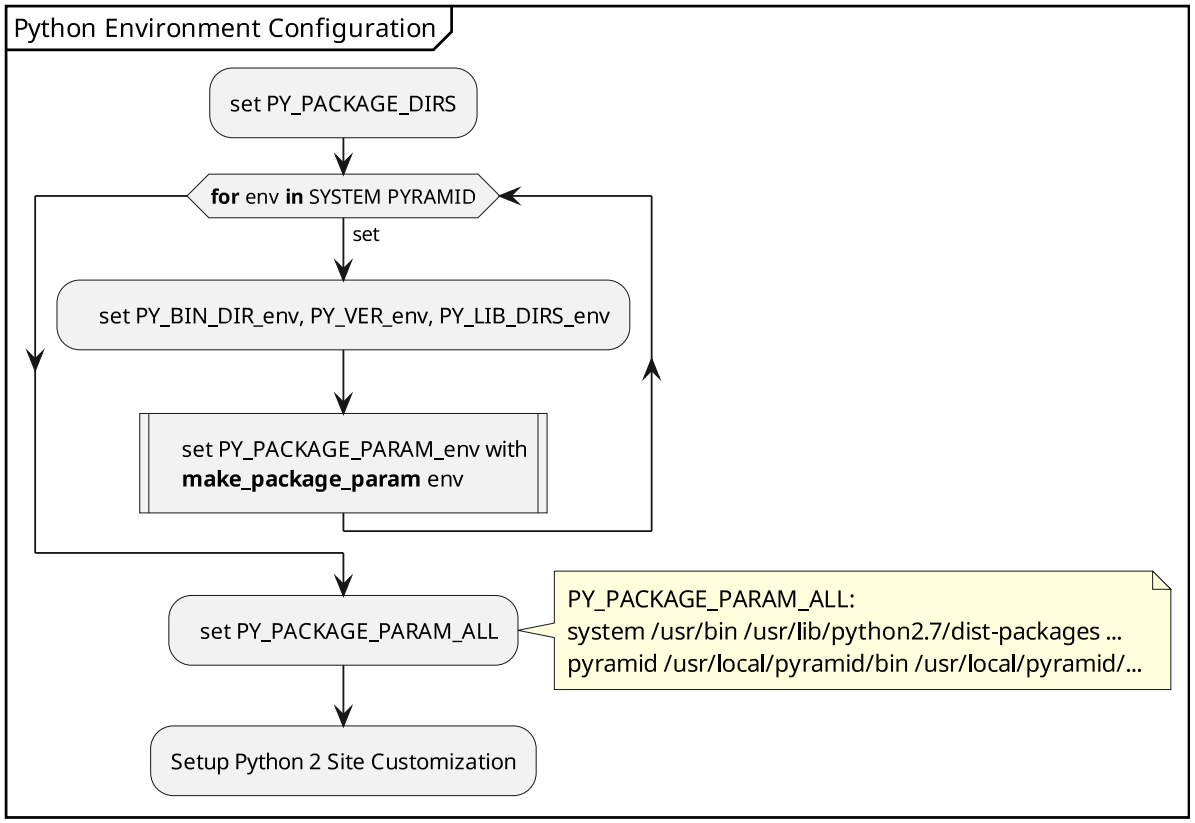


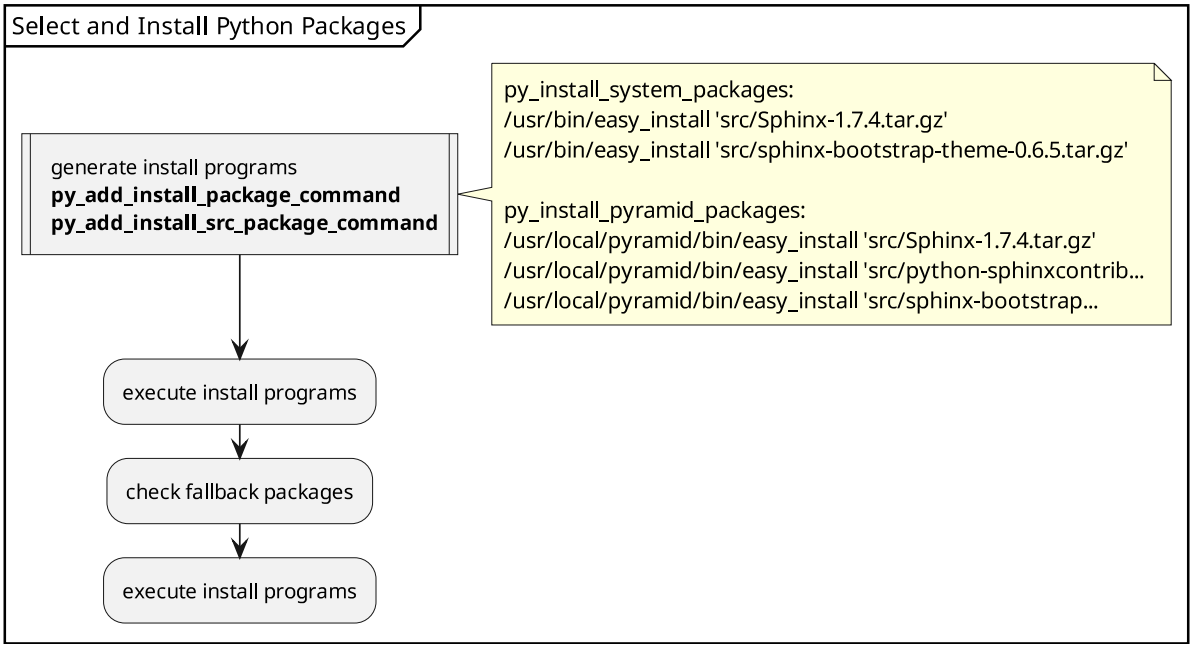
13.4 Activity Diagrams for `bin/inst.sh`

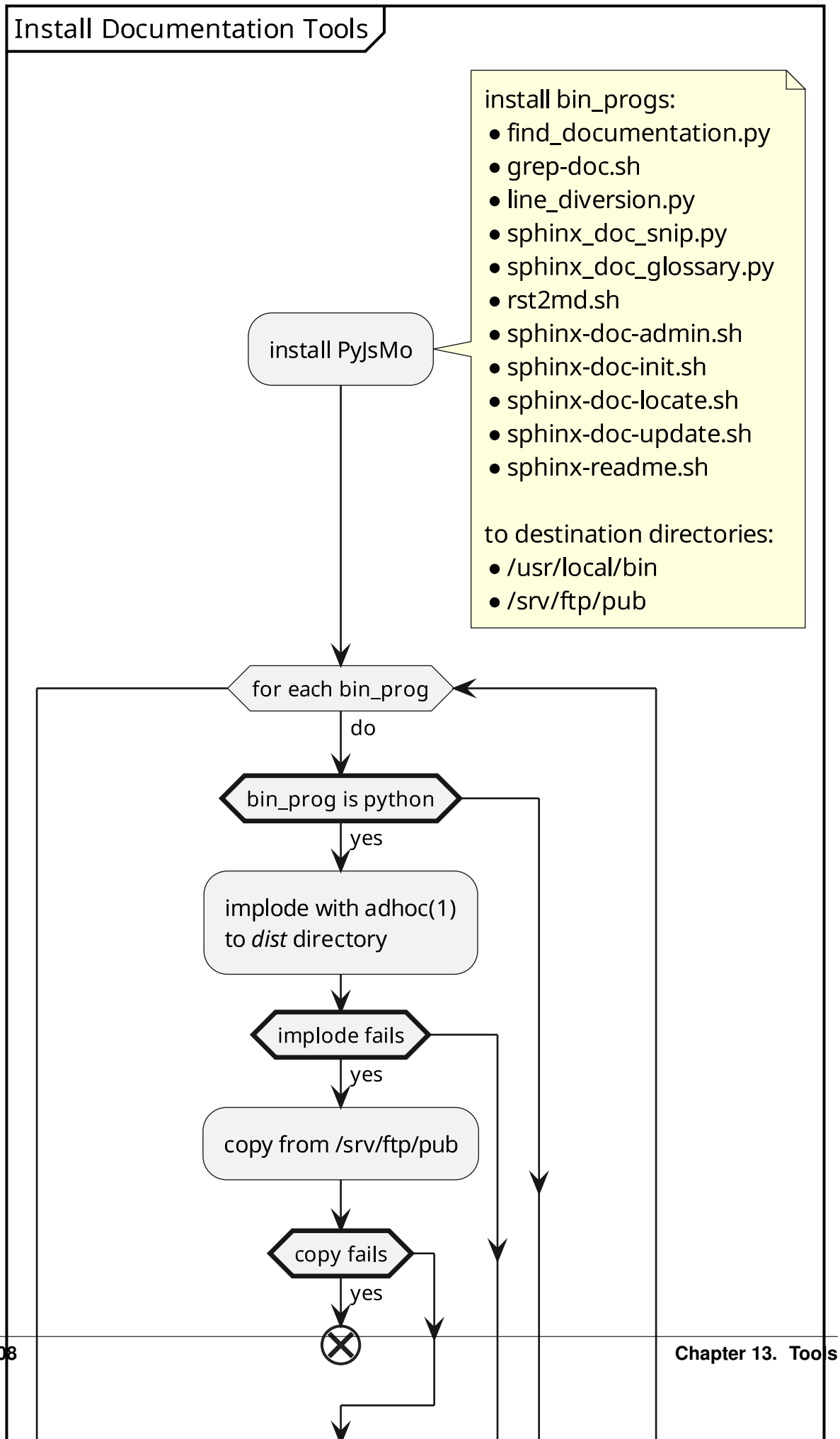



Install System Packages
setup needed and wanted system packages
show host info
setup ws-local repository
determine missing packages
install missing packages


Extra Application Setup
PlantUML update
Install webvector (CSSBox)


Python Environment Configuration
set PY_PACKAGE_DIRS
for env in SYSTEM PYRAMID
set
set PY_BIN_DIR_env, PY_VER_env, PY_LIB_DIRS_env
set PY_PACKAGE_PARAM_env with
make_package_param env
set PY_PACKAGE_PARAM_ALL
PY_PACKAGE_PARAM_ALL:
system /usr/bin /usr/lib/python2.7/dist-packages ...
pyramid /usr/local/pyramid/bin /usr/local/pyramid/...
Setup Python 2 Site Customization


Select and Install Python Packages
generate install programs
py_add_install_package_command
py_add_install_src_package_command
py_install_system_packages:
/usr/bin/easy_install 'src/Sphinx-1.7.4.tar.gz'
/usr/bin/easy_install 'src/sphinx-bootstrap-theme-0.6.5.tar.gz'
py_install_pyramid_packages:
/usr/local/pyramid/bin/easy_install 'src/Sphinx-1.7.4.tar.gz'
/usr/local/pyramid/bin/easy_install 'src/python-sphinxcontrib...
/usr/local/pyramid/bin/easy_install 'src/sphinx-bootstrap...
execute install programs
check fallback packages
execute install programs


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
install PyJsMo
for each bin_prog
do
bin_prog is python
yes
implode with adhoc(1)
to dist directory
implode fails
yes
copy from /srv/ftp/pub
copy fails
yes

CHAPTER

FOURTEEN

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 defintions 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{pmboxdraw}
\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

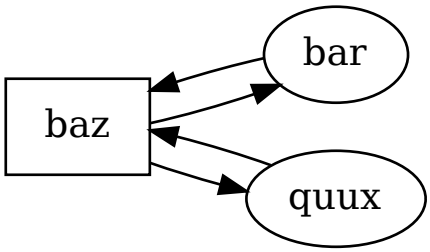
```
apt-get install python-sphinx-rtd-theme
easy_integration git://github.com/snide/sphinx_rtd_theme

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:
```

CHAPTER

FIFTEEN

DIAGRAM GENERATORS

CHAPTER

SIXTEEN

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:kelleyk/emacs

apt-get update && \
apt-get install emacs26 emacs26-el && \
update-alternatives --set emacs /usr/bin/emacs26
```

16.1.1 Setup for emacsen-common

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

```
touch /var/lib/emacsen-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 autoload-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)

(continued from previous page)

```
as a list.")

(defun package--description-file (dir)
  (concat (let ((subdir (file-name-nondirectory
                         (directory-file-name dir))))
            (if (string-match "\\([^.].*?\\)-\\([0-9]+\\(?:[.][0-9]+\\|\\(?
↪:pre\\|beta\\|alpha\\)[0-9]+\\)*\\)" subdir)
                (match-string 1 subdir) subdir))
          "-pkg.el"))

(defconst debian-emacs-flavor 'emacs26
  "A symbol representing the particular debian flavor of emacs running.
Something like 'emacs20, 'xemacs20, etc.")

(if (fboundp 'normal-top-level-add-subdirs-to-load-path)
    (normal-top-level-add-subdirs-to-load-path))

(let ((local-site-lisp "/usr/local/share/emacs/26.3/site-lisp/"))
  (make-directory local-site-lisp t)
  (if (not (member local-site-lisp load-path))
      (setq load-path (cons local-site-lisp load-path))))
EOF
;; |:here:| `
```

Now add the debian specific startup to `/usr/share/emacs26/site-lisp/site-start.el`:

```
;; |:here:|
cat <<'EOF' >/usr/share/emacs26/site-lisp/site-start.el
(if (load "debian-startup" t t nil)
    (debian-startup debian-emacs-flavor))
EOF
;; |:here:| `
```

16.1.2 Obsolete packages

Remove these packages, since they do not work:

```
apt-get purge prolog-el
apt-get purge dash-el
```

16.1.3 Modify/repair packages

If necessary (Ubuntu 16.04/18.04):

```
grep '(("\\\\\\\\\\[Ss]ubref{\\\\(\[^{}\\n\\r\\\\%,\]\*\\\\)" 1 LaTeX-label-list "}")))' /usr/
↪share/emacs/site-lisp/auctex/style/subfigure.el
```

apply auctex patch manually (obviously this will not work with patch(1), since TABs are lost):

```
--- /usr/share/emacs/site-lisp/auctex/style/subfigure.el-000 2015-11-19 15:26:11.000000000␣
↪+0100
+++ /usr/share/emacs/site-lisp/auctex/style/subfigure.el     2019-09-03 21:03:22.354599330␣
↪+0200
@@ -47,8 +47,8 @@
    ;; Install completion for labels:
    (setq TeX-complete-list
          (append
-       '(("\\\\[Ss]ubref{\\([^{}\n\r\\%,]*\\)" 1 LaTeX-label-list "}")))
-       TeX-complete-list)
+       '(("\\\\[Ss]ubref{\\([^{}\n\r\\%,]*\\)" 1 LaTeX-label-list "}"))
+       TeX-complete-list))

    ;; Fontification
    (when (and (featurep 'font-latex)
```

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 intstalled 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 -1 /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 explicitely 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

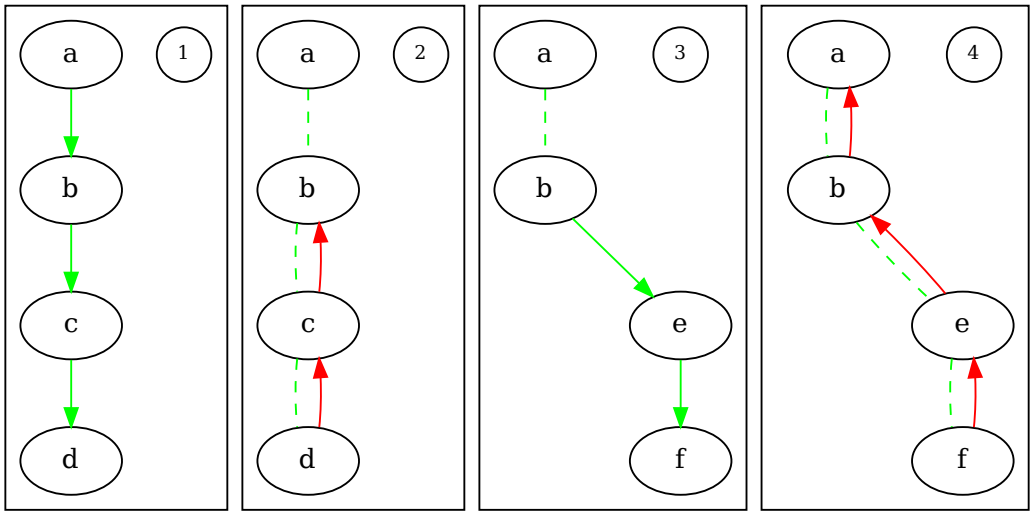
16.3 Undo

- 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.

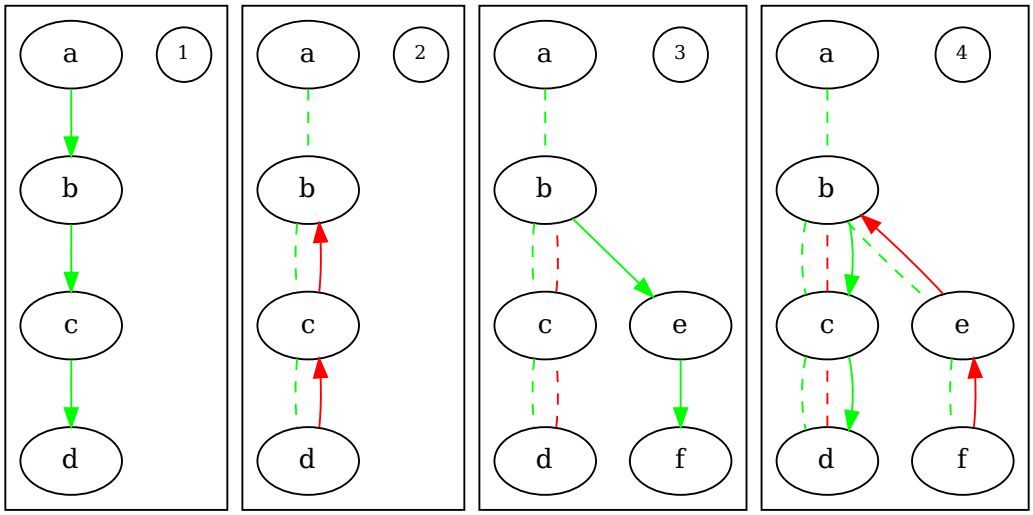
Step	Action	Description
1)	Edit	4 chunks added **a, b, c, d**
2)	Undo	2 chunks removed **d, c**
3)	Edit	2 chunks added **e, f**
4)	Undo	2 chunks removed **f, e**
	Undo	1 chunk removed **b**



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 .

`C-/`	undo-tree-undo
`C-_`	undo-tree-undo
`C-?`	undo-tree-redo
`C-x u`	undo-tree-visualize

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:

dy**namic**
dy**lan**
dy**l**
dy**lan-console**
dy**lan-repl**
dy**lanlid**
dy**lan-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 to 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 verfied, 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	Projekt-code	Farbe	Enddatum bestätigt	Plan-Enddatum	Fortschritt	PSP	Status	Projekt-status	erledigt	geschlossen	Hyperlink
1	PROJECT-LEAVE-PERIOD	Administrative						0	1	recorded		0	0	http://scherer.wiedenmann.intern/projeqtor/view/main.php?directAccess=true&objectClass=Project&objectId=1
2	Traverse1	Fixed Price	Kunde1	123456		2020-01-23	2020-01-24	82	2	done	secured	0	0	http://scherer.wiedenmann.intern/projeqtor/view/main.php?directAccess=true&objectClass=Project&objectId=2
3	Traversenhalterung	Fixed Price	Kunde1			2019-11-16	2020-01-09	100	2.2	done	secured	1	0	http://scherer.wiedenmann.intern/projeqtor/view/main.php?directAccess=true&objectClass=Project&objectId=3
4	Meues Projekt	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	Projekt_1	Fixed Price						0	4	recorded		0	0	http://scherer.wiedenmann.intern/projeqtor/view/main.php?directAccess=true&objectClass=Project&objectId=5
6	Projekt_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 alskfjjkl asdflkjaslkfj 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 explicitely.

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 dired-xx-copy-directory-as-kill
C-c u d o	M-x dired-xx-copy-directory-as-kill-other-window
C-c u d w	M-x dired-xx-copy-directory-as-kill-windos
C-c u d f	M-x dired-xx-copy-filename-as-kill

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:

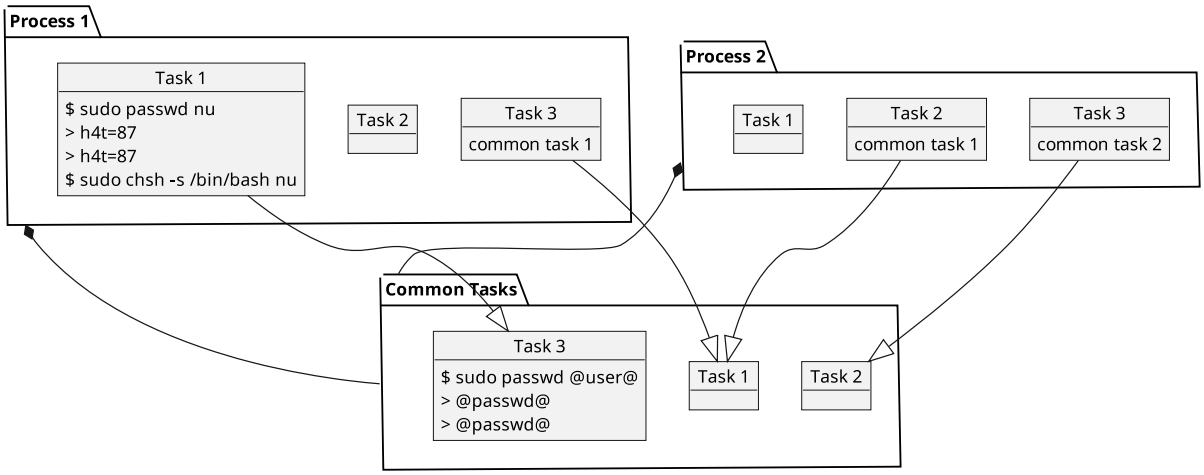
```
`Zonentransfer - Wikipedia <https://de.wikipedia.org/wiki/Zonentransfer>`_
```

CHAPTER

SEVENTEEN

DOCUMENT SNIPPETS

Document snippets solve the problem of describing mutliple 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 explicitely 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 *WHAT=WITH*. *@WHAT@* is replaced by string *WITH* after resolving snippets.
-f, –format FMT	format for output documents. Default: *{dir}/{base}-x{ext}*. Placeholders: *path*, *dir*, *file*, *base*, *ext*, *sbase* (secondary base of *base*), *sext*. 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[=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 *python setup.py install*
–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 defintions 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}{sext}'
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)

(continued from previous 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`",
                    "- |<-sn1p->| see `Unlabeled Section`_",
                    "",
                    "  #. |<-snap->| item 2",
                    "  #. |<-snap->| item 1",
                    "",
                    "  |<-sn1p->| 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: - |<-sn1p->| 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:   |<-sn1p->| 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.

CHAPTER

EIGHTEEN

SNIPPETS

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.

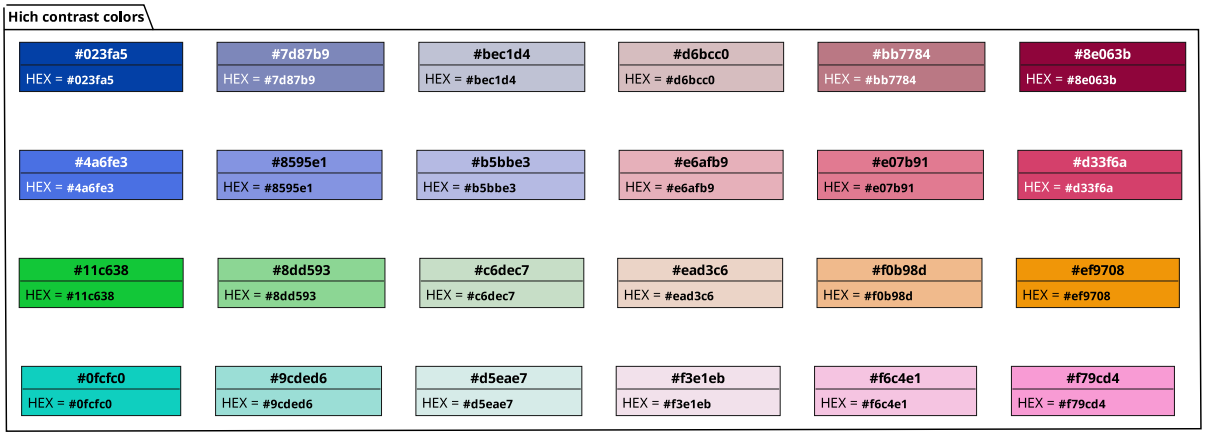
CHAPTER

NINETEEN

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

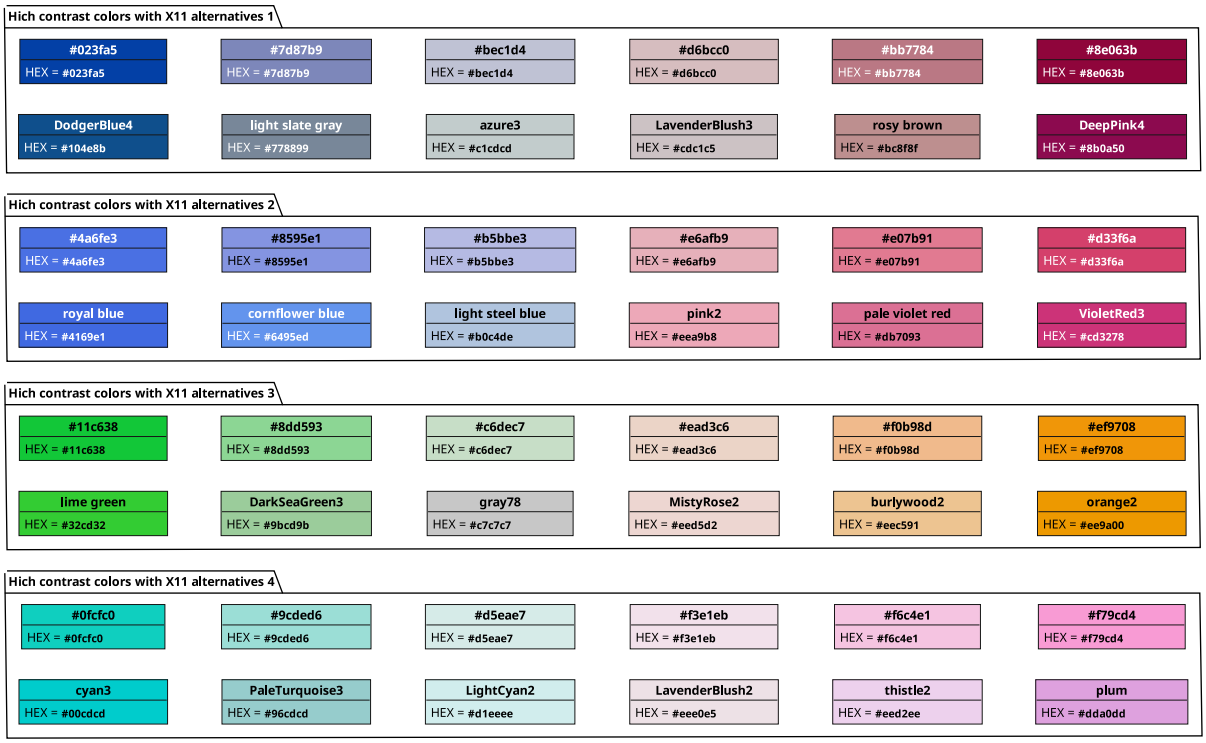


19.2 Thunderbird Tags

Example for Thundebird 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



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

CHAPTER

TWENTY

X11 COLORS

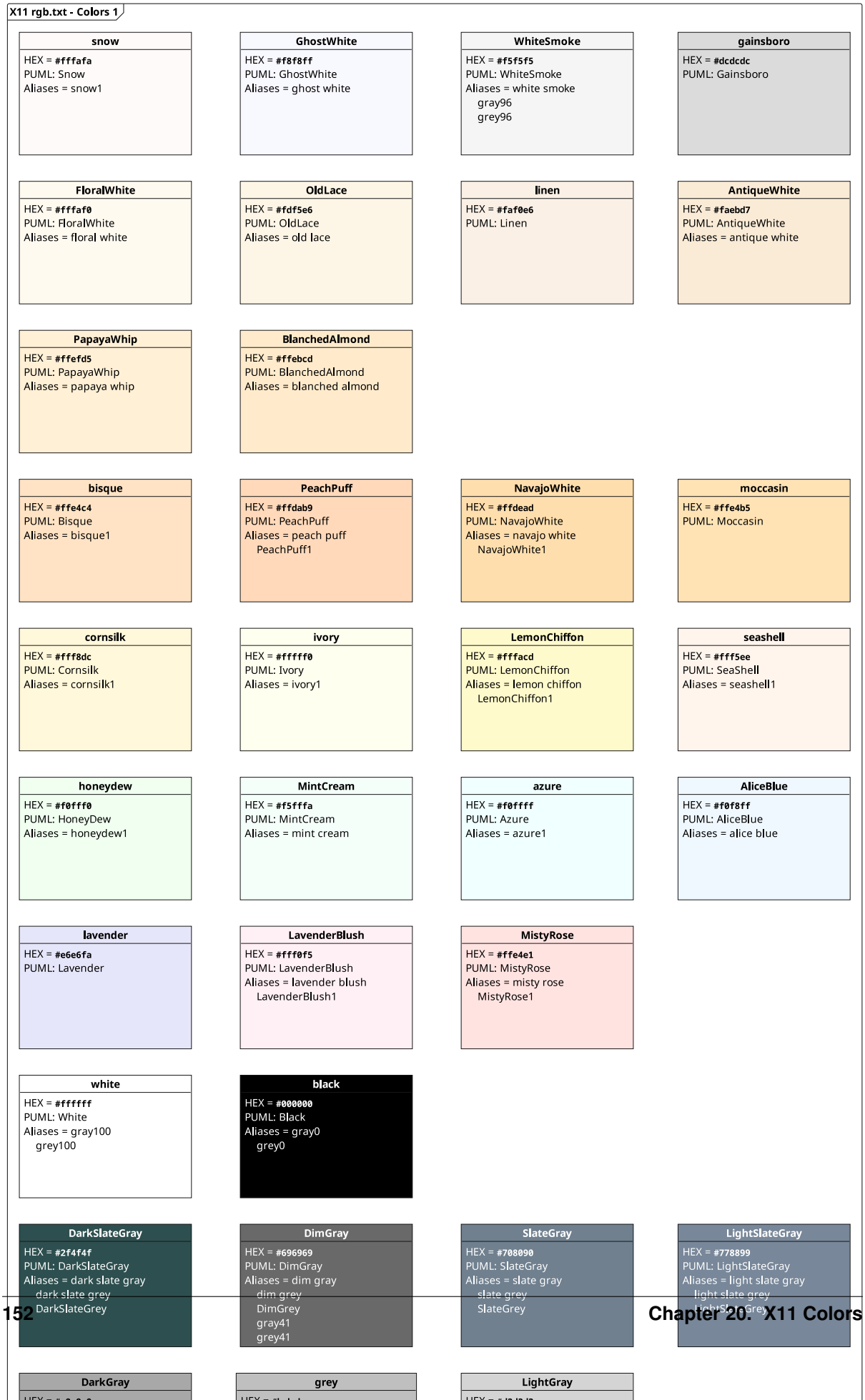
PlantUML follows the web colors, which incorporate a subset of X11 color names. The X11 colors are especially useful to visualize color ranges.

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

20.2 X11 Colors




X11 rgb.txt - Colors 2
MidnightBlue
HEX = #191970
PUML: MidnightBlue
Aliases = midnight blue
NavyBlue
HEX = #000080
PUML: Navy
Aliases = navy
navy blue
CornflowerBlue
HEX = #6495ed
PUML: CornflowerBlue
Aliases = cornflower blue
DarkSlateBlue
HEX = #483d8b
PUML: DarkSlateBlue
Aliases = dark slate blue
SlateBlue
HEX = #6a5acd
PUML: SlateBlue
Aliases = slate blue
MediumSlateBlue
HEX = #7b68ee
PUML: MediumSlateBlue
Aliases = medium slate blue
LightSlateBlue
HEX = #8470ff
PUML: #8470ff
Aliases = light slate blue
DarkBlue
HEX = #00008b
PUML: DarkBlue
Aliases = blue4
dark blue
MediumBlue
HEX = #0000cd
PUML: MediumBlue
Aliases = medium blue
blue3
RoyalBlue
HEX = #4169e1
PUML: RoyalBlue
Aliases = royal blue
blue
HEX = #0000ff
PUML: Blue
Aliases = blue1
DodgerBlue
HEX = #1e90ff
PUML: DodgerBlue
Aliases = dodger blue
DodgerBlue1
DeepSkyBlue
HEX = #00bfff
PUML: DeepSkyBlue
Aliases = deep sky blue
DeepSkyBlue1
SkyBlue
HEX = #87ceeb
PUML: SkyBlue
Aliases = sky blue
LightSkyBlue
HEX = #87cefa
PUML: LightSkyBlue
Aliases = light sky blue
SteelBlue
HEX = #4682b4
PUML: SteelBlue
Aliases = steel blue
LightSteelBlue
HEX = #b0c4de
PUML: LightSteelBlue
Aliases = light steel blue
LightBlue
HEX = #add8e6
PUML: LightBlue
Aliases = light blue
PowderBlue
HEX = #b0e0e6
PUML: PowderBlue
Aliases = powder blue
PaleTurquoise
HEX = #afeeee
PUML: PaleTurquoise
Aliases = pale turquoise
DarkTurquoise
HEX = #00ced1
PUML: DarkTurquoise
Aliases = dark turquoise
MediumTurquoise
HEX = #48d1cc
PUML: MediumTurquoise
Aliases = medium turquoise
turquoise
HEX = #40e0d0
PUML: Turquoise
DarkCyan
HEX = #008b8b
PUML: DarkCyan
Aliases = cyan4
dark cyan
cyan
HEX = #00ffff
PUML: Aqua
Aliases = cyan1
LightCyan
HEX = #e0ffff
PUML: LightCyan
Aliases = light cyan
LightCyan1
CadetBlue
HEX = #5f9ea0
PUML: CadetBlue
Aliases = cadet blue
MediumAquamarine
HEX = #66cdaa
PUML: MediumAquaMarine
Aliases = medium aquamarine
aquamarine3
aquamarine
HEX = #7fffd4
PUML: Aquamarine
Aliases = aquamarine1


X11 rgb.txt - Colors 3
DarkGreen
HEX = #006400
PUML: DarkGreen
Aliases = dark green
DarkOliveGreen
HEX = #556b2f
PUML: DarkOliveGreen
Aliases = dark olive green
DarkSeaGreen
HEX = #8fbc8f
PUML: DarkSeaGreen
Aliases = dark sea green
SeaGreen
HEX = #2e8b57
PUML: SeaGreen
Aliases = sea green
SeaGreen4
MediumSeaGreen
HEX = #3cb371
PUML: MediumSeaGreen
Aliases = medium sea green
LightSeaGreen
HEX = #20b2aa
PUML: LightSeaGreen
Aliases = light sea green
LightGreen
HEX = #90ee90
PUML: LightGreen
Aliases = PaleGreen2
light green
PaleGreen
HEX = #98fb98
PUML: PaleGreen
Aliases = pale green
SpringGreen
HEX = #00ff7f
PUML: SpringGreen
Aliases = spring green
SpringGreen1
LawnGreen
HEX = #7cfc00
PUML: LawnGreen
Aliases = lawn green
green
HEX = #00ff00
PUML: Lime
Aliases = green1
chartreuse
HEX = #7fff00
PUML: Chartreuse
Aliases = chartreuse1
MediumSpringGreen
HEX = #00fa9a
PUML: MediumSpringGreen
Aliases = medium spring green
GreenYellow
HEX = #adff2f
PUML: GreenYellow
Aliases = green yellow
LimeGreen
HEX = #32cd32
PUML: LimeGreen
Aliases = lime green
YellowGreen
HEX = #9acd32
PUML: YellowGreen
Aliases = yellow green
OliveDrab3
ForestGreen
HEX = #228b22
PUML: ForestGreen
Aliases = forest green
OliveDrab
HEX = #6b8e23
PUML: OliveDrab
Aliases = olive drab
DarkKhaki
HEX = #bdb76b
PUML: DarkKhaki
Aliases = dark khaki
khaki
HEX = #f0e68c
PUML: Khaki
PaleGoldenrod
HEX = #eee8aa
PUML: PaleGoldenRod
Aliases = pale goldenrod
LightGoldenrodYellow
HEX = #fafad2
PUML: LightGoldenRodYellow
Aliases = light goldenrod yellow
LightYellow
HEX = #ffffe0
PUML: LightYellow
Aliases = light yellow
LightYellow1
yellow
HEX = #ffff00
PUML: Yellow
Aliases = yellow1
gold
HEX = #ffd700
PUML: Gold
Aliases = gold1
LightGoldenrod
HEX = #eedd82
PUML: #eedd82
Aliases = light goldenrod
goldenrod
HEX = #daa520
PUML: GoldenRod
DarkGoldenrod
HEX = #b8860b
PUML: DarkGoldenRod
Aliases = dark goldenrod
RosyBrown
HEX = #bc8f8f
PUML: RosyBrown
Aliases = rosy brown
IndianRed
HEX = #cd5c5c
PUML: IndianRed
Aliases = indian red
SaddleBrown
HEX = #8b4513
PUML: SaddleBrown
Aliases = saddle brown
chocolate4
sienna
HEX = #a0522d
PUML: Sienna
peru
HEX = #cd853f
PUML: Peru
Aliases = tan3
burlywood
HEX = #deb887
PUML: BurlyWood
beige
HEX = #f5f5dc
PUML: Beige
wheat
HEX = #f5deb3
PUML: Wheat
SandyBrown
HEX = #f4a460
PUML: SandyBrown
Aliases = sandy brown
tan
HEX = #d2b48c
PUML: Tan
chocolate
HEX = #d2691e
PUML: Chocolate
FireBrick
HEX = #b22222
PUML: FireBrick

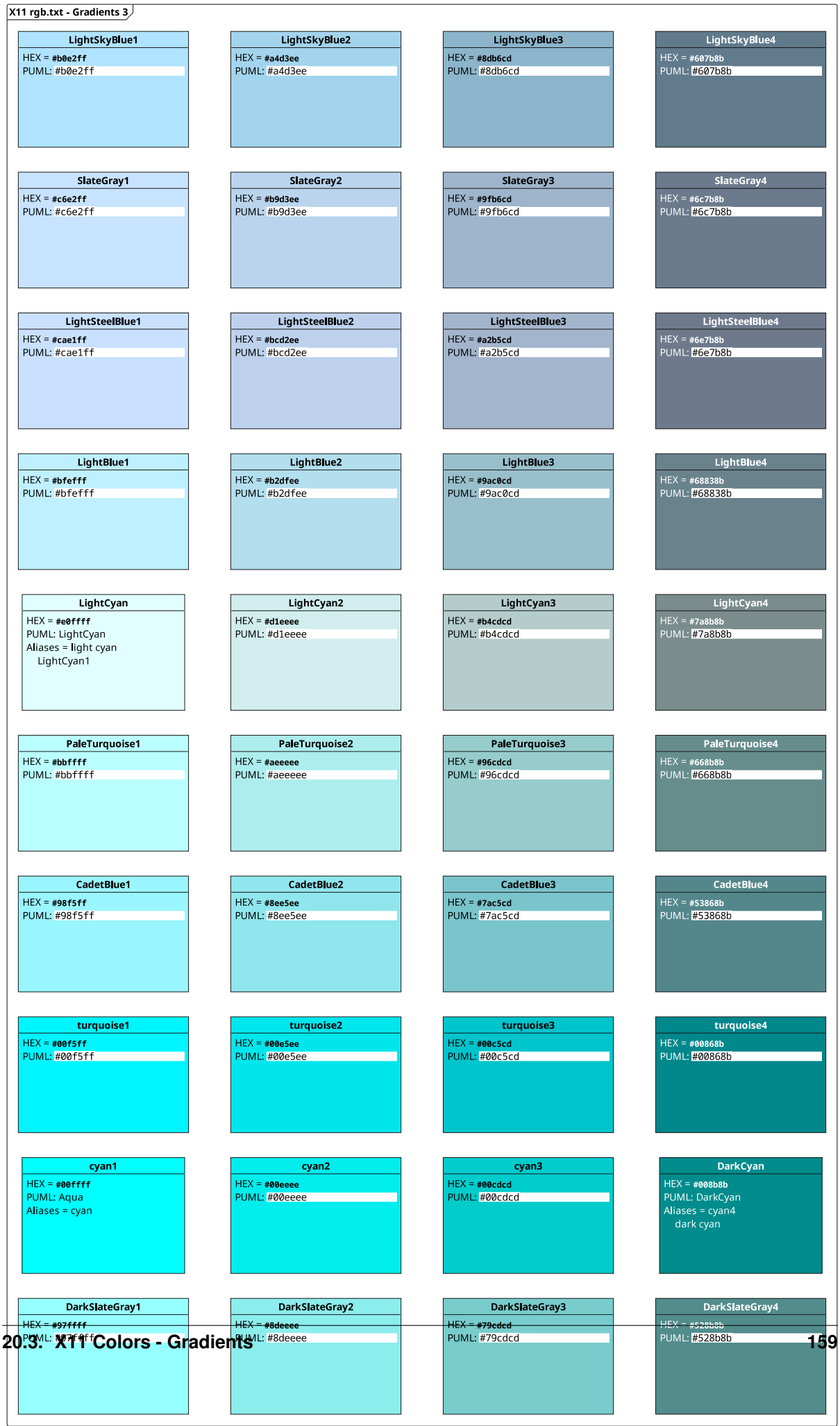

X11 rgb.txt - Colors 4
orange
HEX = #ffa500
PUML: Orange
Aliases = orange1
DarkOrange
HEX = #ff8c00
PUML: Darkorange
Aliases = dark orange
coral
HEX = #ff7f50
PUML: Coral
LightCoral
HEX = #f08080
PUML: LightCoral
Aliases = light coral
tomato
HEX = #ff6347
PUML: Tomato
Aliases = tomato1
OrangeRed
HEX = #ff4500
PUML: OrangeRed
Aliases = orange red
OrangeRed1
red
HEX = #ff0000
PUML: Red
Aliases = red1
DarkRed
HEX = #8b0000
PUML: DarkRed
Aliases = red4
dark red
HotPink
HEX = #ff69b4
PUML: HotPink
Aliases = hot pink
DeepPink
HEX = #ff1493
PUML: DeepPink
Aliases = deep pink
DeepPink1
pink
HEX = #ffc0cb
PUML: Pink
LightPink
HEX = #ffb6c1
PUML: LightPink
Aliases = light pink
PaleVioletRed
HEX = #db7093
PUML: #db7093
Aliases = pale violet red
maroon
HEX = #b03060
PUML: #b03060
MediumVioletRed
HEX = #c71585
PUML: MediumVioletRed
Aliases = medium violet red
VioletRed
HEX = #d02090
PUML: #d02090
Aliases = violet red
DarkMagenta
HEX = #8b008b
PUML: DarkMagenta
Aliases = magenta4
dark magenta
magenta
HEX = #ff00ff
PUML: Fuchsia
Aliases = magenta1
violet
HEX = #ee82ee
PUML: Violet
plum
HEX = #dda0dd
PUML: Plum
orchid
HEX = #da70d6
PUML: Orchid
MediumOrchid
HEX = #ba55d3
PUML: MediumOrchid
Aliases = medium orchid
DarkOrchid
HEX = #9932cc
PUML: DarkOrchid
Aliases = dark orchid
DarkViolet
HEX = #9400d3
PUML: DarkViolet
Aliases = dark violet
BlueViolet
HEX = #8a2be2
PUML: BlueViolet
Aliases = blue violet
purple
HEX = #a020f0
PUML: #a020f0
MediumPurple
HEX = #9370db
PUML: #9370db
Aliases = medium purple
thistle
HEX = #d8bfd8
PUML: Thistle

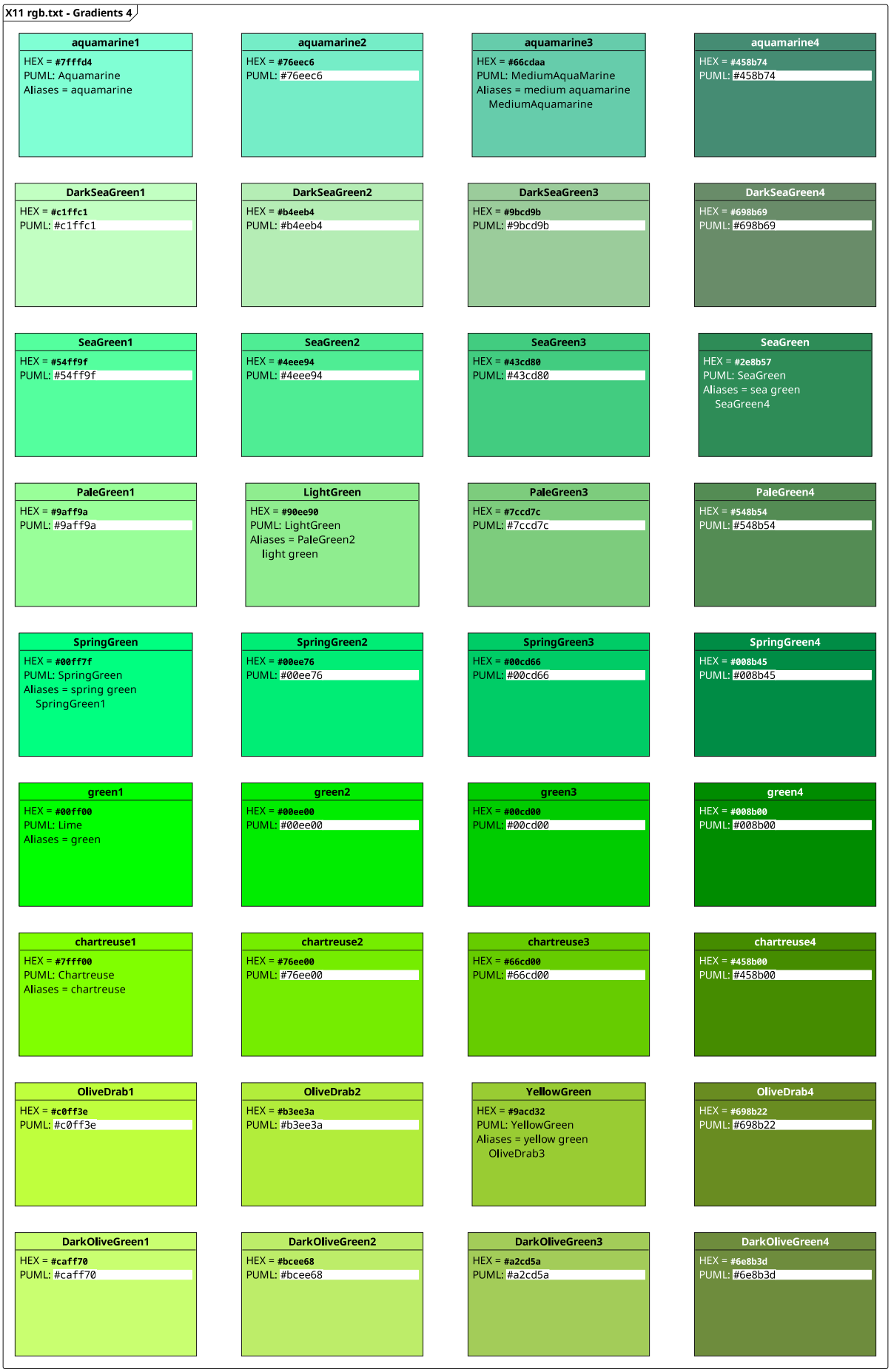
20.3 X11 Colors - Gradients

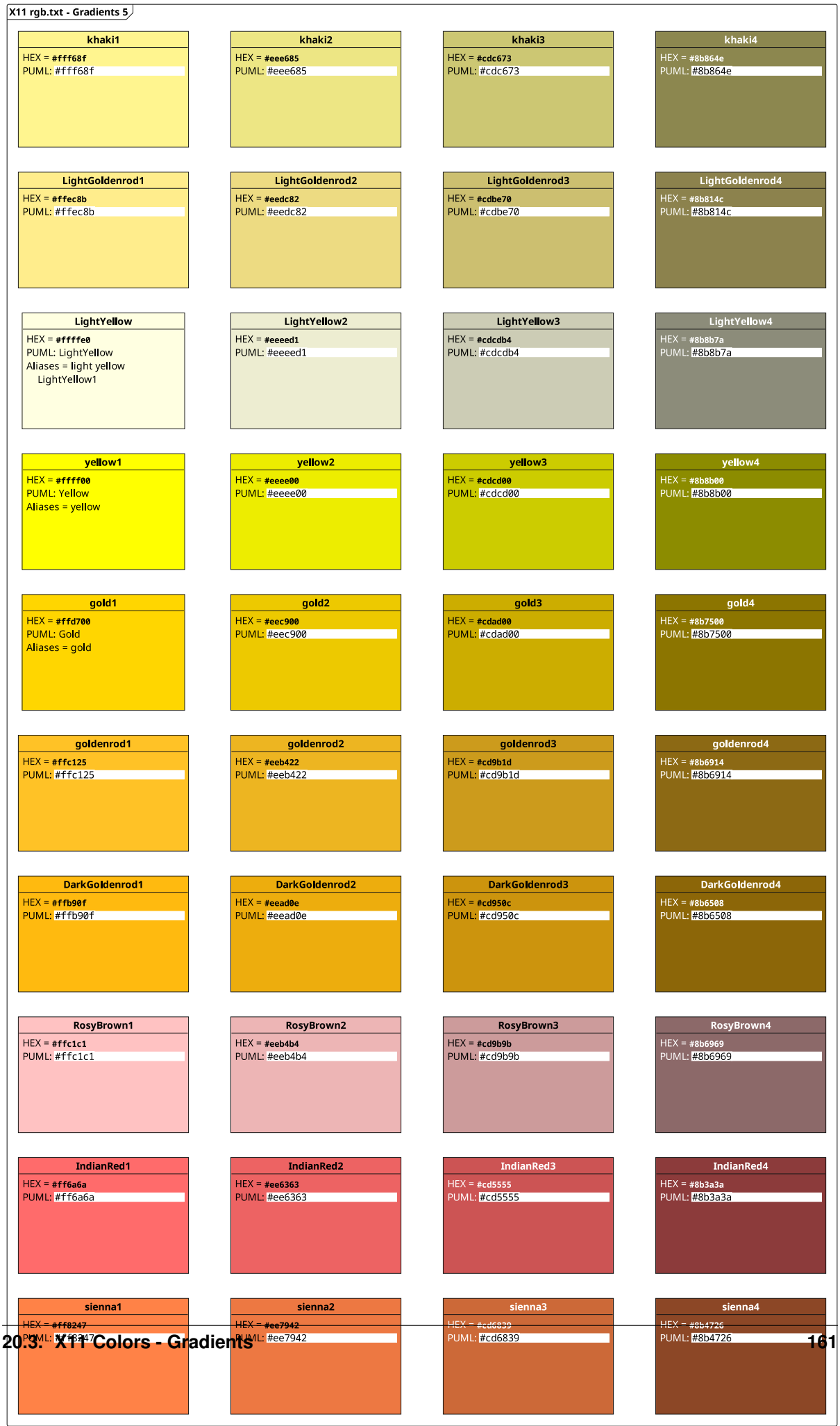
X11 rgb.txt - Gradients 1

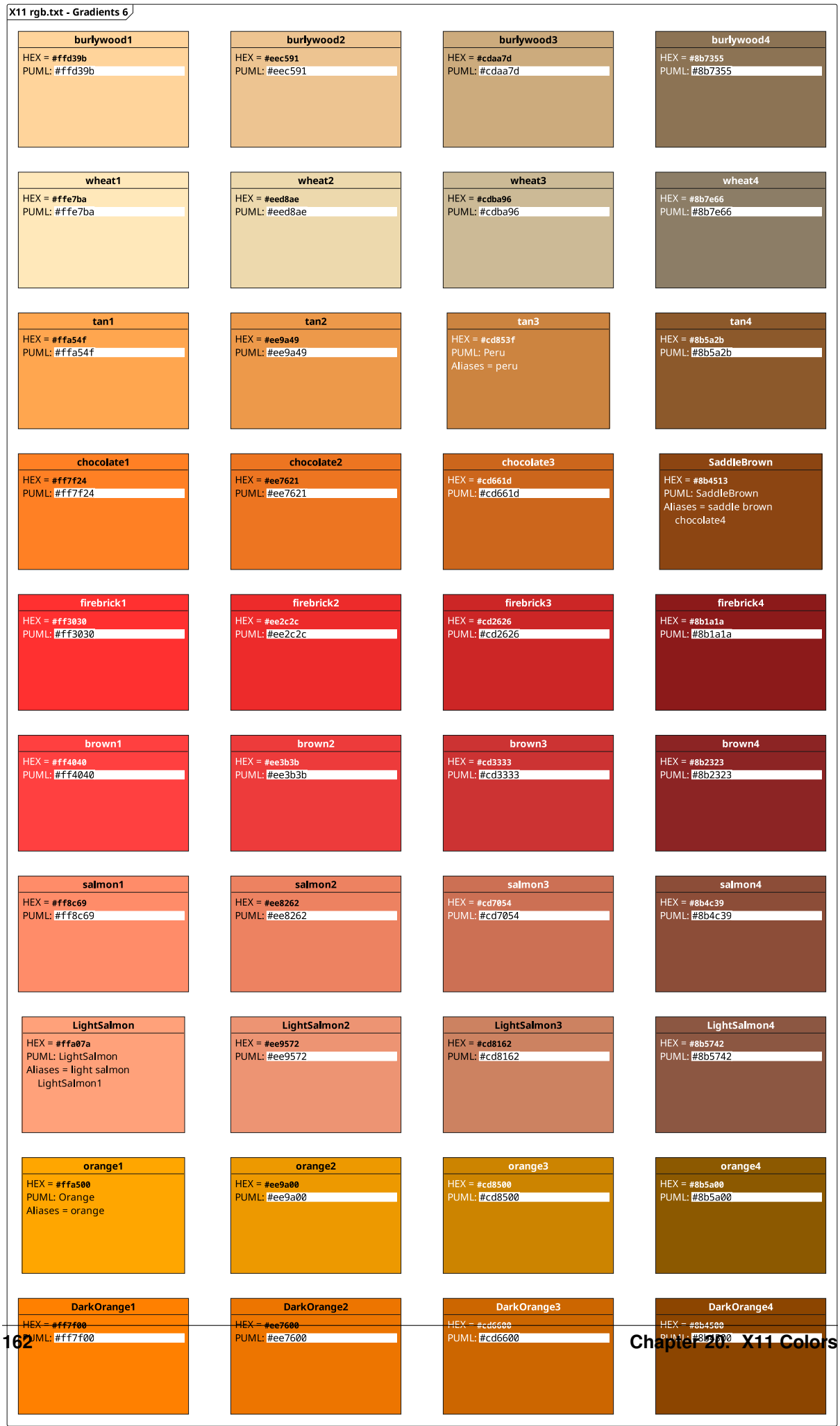
snow1 HEX = **#fffafa** PUML: Snow Aliases = snow	**snow2** HEX = **#eee9e9** PUML: #eee9e9	**snow3** HEX = **#cdc9c9** PUML: #cdc9c9	**snow4** HEX = **#8b8989** PUML: #8b8989
seashell1 HEX = **#fff5ee** PUML: SeaShell Aliases = seashell	**seashell2** HEX = **#eee5de** PUML: #eee5de	**seashell3** HEX = **#cdc5bf** PUML: #cdc5bf	**seashell4** HEX = **#8b8682** PUML: #8b8682
AntiqueWhite1 HEX = **#ffefdb** PUML: #ffefdb	**AntiqueWhite2** HEX = **#eedfcc** PUML: #eedfcc	**AntiqueWhite3** HEX = **#cdc0b0** PUML: #cdc0b0	**AntiqueWhite4** HEX = **#8b8378** PUML: #8b8378
bisque1 HEX = **#ffe4c4** PUML: Bisque Aliases = bisque	**bisque2** HEX = **#eed5b7** PUML: #eed5b7	**bisque3** HEX = **#cdb79e** PUML: #cdb79e	**bisque4** HEX = **#8b7d6b** PUML: #8b7d6b
PeachPuff HEX = **#ffdab9** PUML: PeachPuff Aliases = peach puff PeachPuff1	**PeachPuff2** HEX = **#eecbad** PUML: #eecbad	**PeachPuff3** HEX = **#cdaf95** PUML: #cdaf95	**PeachPuff4** HEX = **#8b7765** PUML: #8b7765
NavajoWhite HEX = **#ffdead** PUML: NavajoWhite Aliases = navajo white NavajoWhite1	**NavajoWhite2** HEX = **#eecfa1** PUML: #eecfa1	**NavajoWhite3** HEX = **#cdb38b** PUML: #cdb38b	**NavajoWhite4** HEX = **#8b795e** PUML: #8b795e
LemonChiffon HEX = **#fffacd** PUML: LemonChiffon Aliases = lemon chiffon LemonChiffon1	**LemonChiffon2** HEX = **#eee9bf** PUML: #eee9bf	**LemonChiffon3** HEX = **#cdc9a5** PUML: #cdc9a5	**LemonChiffon4** HEX = **#8b8970** PUML: #8b8970
cornsilk1 HEX = **#fff8dc** PUML: Cornsilk Aliases = cornsilk	**cornsilk2** HEX = **#eee8cd** PUML: #eee8cd	**cornsilk3** HEX = **#cdc8b1** PUML: #cdc8b1	**cornsilk4** HEX = **#8b8878** PUML: #8b8878
ivory1 HEX = **#fffff0** PUML: Ivory Aliases = ivory	**ivory2** HEX = **#eeeee0** PUML: #eeeee0	**ivory3** HEX = **#cdcdc1** PUML: #cdcdc1	**ivory4** HEX = **#8b8b83** PUML: #8b8b83
honeydew1 HEX = **#f0fff0** PUML: HoneyDew Aliases = honeydew	**honeydew2** HEX = **#e0eee0** PUML: #e0eee0	**honeydew3** HEX = **#c1cdc1** PUML: #c1cdc1	**honeydew4** HEX = **#838b83** PUML: #838b83


X11 rgb.txt - Gradients 2
LavenderBlush
HEX = #fff0f5
PUML: LavenderBlush
Aliases = lavender blush
LavenderBlush1
LavenderBlush2
HEX = #eee0e5
PUML: #eee0e5
LavenderBlush3
HEX = #cdc1c5
PUML: #cdc1c5
LavenderBlush4
HEX = #8b8386
PUML: #8b8386
MistyRose
HEX = #ffe4e1
PUML: MistyRose
Aliases = misty rose
MistyRose1
MistyRose2
HEX = #eed5d2
PUML: #eed5d2
MistyRose3
HEX = #cdb7b5
PUML: #cdb7b5
MistyRose4
HEX = #8b7d7b
PUML: #8b7d7b
azure1
HEX = #f0ffff
PUML: Azure
Aliases = azure
azure2
HEX = #e0eeee
PUML: #e0eeee
azure3
HEX = #c1cdcd
PUML: #c1cdcd
azure4
HEX = #838b8b
PUML: #838b8b
SlateBlue1
HEX = #836fff
PUML: #836fff
SlateBlue2
HEX = #7a67ee
PUML: #7a67ee
SlateBlue3
HEX = #6959cd
PUML: #6959cd
SlateBlue4
HEX = #473c8b
PUML: #473c8b
RoyalBlue1
HEX = #4876ff
PUML: #4876ff
RoyalBlue2
HEX = #436eee
PUML: #436eee
RoyalBlue3
HEX = #3a5fcd
PUML: #3a5fcd
RoyalBlue4
HEX = #27408b
PUML: #27408b
blue1
HEX = #0000ff
PUML: Blue
Aliases = blue
blue2
HEX = #0000ee
PUML: #0000ee
MediumBlue
HEX = #0000cd
PUML: MediumBlue
Aliases = medium blue
blue3
DarkBlue
HEX = #00008b
PUML: DarkBlue
Aliases = blue4
dark blue
DodgerBlue
HEX = #1e90ff
PUML: DodgerBlue
Aliases = dodger blue
DodgerBlue1
DodgerBlue2
HEX = #1c86ee
PUML: #1c86ee
DodgerBlue3
HEX = #1874cd
PUML: #1874cd
DodgerBlue4
HEX = #104e8b
PUML: #104e8b
SteelBlue1
HEX = #63b8ff
PUML: #63b8ff
SteelBlue2
HEX = #5cacee
PUML: #5cacee
SteelBlue3
HEX = #4f94cd
PUML: #4f94cd
SteelBlue4
HEX = #36648b
PUML: #36648b
DeepSkyBlue
HEX = #00bfff
PUML: DeepSkyBlue
Aliases = deep sky blue
DeepSkyBlue1
DeepSkyBlue2
HEX = #00b2ee
PUML: #00b2ee
DeepSkyBlue3
HEX = #009acd
PUML: #009acd
DeepSkyBlue4
HEX = #00688b
PUML: #00688b
SkyBlue1
HEX = #87ceff
PUML: #87ceff
SkyBlue2
HEX = #7ec0ee
PUML: #7ec0ee
SkyBlue3
HEX = #6ca6cd
PUML: #6ca6cd
SkyBlue4


X11 rgb.txt - Gradients 3
LightSkyBlue1
HEX = #b0e2ff
PUML: #b0e2ff
LightSkyBlue2
HEX = #a4d3ee
PUML: #a4d3ee
LightSkyBlue3
HEX = #8db6cd
PUML: #8db6cd
LightSkyBlue4
HEX = #607b8b
PUML: #607b8b
SlateGray1
HEX = #c6e2ff
PUML: #c6e2ff
SlateGray2
HEX = #b9d3ee
PUML: #b9d3ee
SlateGray3
HEX = #9fb6cd
PUML: #9fb6cd
SlateGray4
HEX = #6c7b8b
PUML: #6c7b8b
LightSteelBlue1
HEX = #cae1ff
PUML: #cae1ff
LightSteelBlue2
HEX = #bcd2ee
PUML: #bcd2ee
LightSteelBlue3
HEX = #a2b5cd
PUML: #a2b5cd
LightSteelBlue4
HEX = #6e7b8b
PUML: #6e7b8b
LightBlue1
HEX = #bfefff
PUML: #bfefff
LightBlue2
HEX = #b2dfee
PUML: #b2dfee
LightBlue3
HEX = #9ac0cd
PUML: #9ac0cd
LightBlue4
HEX = #68838b
PUML: #68838b
LightCyan
HEX = #e0ffff
PUML: LightCyan
Aliases = light cyan
LightCyan1
LightCyan2
HEX = #d1eeee
PUML: #d1eeee
LightCyan3
HEX = #b4cdcd
PUML: #b4cdcd
LightCyan4
HEX = #7a8b8b
PUML: #7a8b8b
PaleTurquoise1
HEX = #bbffff
PUML: #bbffff
PaleTurquoise2
HEX = #aeeeee
PUML: #aeeeee
PaleTurquoise3
HEX = #96cdcd
PUML: #96cdcd
PaleTurquoise4
HEX = #668b8b
PUML: #668b8b
CadetBlue1
HEX = #98f5ff
PUML: #98f5ff
CadetBlue2
HEX = #8ee5ee
PUML: #8ee5ee
CadetBlue3
HEX = #7ac5cd
PUML: #7ac5cd
CadetBlue4
HEX = #53868b
PUML: #53868b
turquoise1
HEX = #00f5ff
PUML: #00f5ff
turquoise2
HEX = #00e5ee
PUML: #00e5ee
turquoise3
HEX = #00c5cd
PUML: #00c5cd
turquoise4
HEX = #00868b
PUML: #00868b
cyan1
HEX = #00ffff
PUML: Aqua
Aliases = cyan
cyan2
HEX = #00eeee
PUML: #00eeee
cyan3
HEX = #00cdcd
PUML: #00cdcd
DarkCyan
HEX = #008b8b
PUML: DarkCyan
Aliases = cyan4
dark cyan
DarkSlateGray1
HEX = #97ffff
DarkSlateGray2
HEX = #8deeee
PUML: #8deeee
DarkSlateGray3
HEX = #79cdcd
PUML: #79cdcd
DarkSlateGray4
PUML: #528b8b


X11 rgb.txt - Gradients 4
aquamarine1
HEX = #7fffd4
PUML: Aquamarine
Aliases = aquamarine
aquamarine2
HEX = #76eec6
PUML: #76eec6
aquamarine3
HEX = #66cdaa
PUML: MediumAquaMarine
Aliases = medium aquamarine
MediumAquamarine
aquamarine4
HEX = #458b74
PUML: #458b74
DarkSeaGreen1
HEX = #c1ffc1
PUML: #c1ffc1
DarkSeaGreen2
HEX = #b4eeb4
PUML: #b4eeb4
DarkSeaGreen3
HEX = #9bcd9b
PUML: #9bcd9b
DarkSeaGreen4
HEX = #698b69
PUML: #698b69
SeaGreen1
HEX = #54ff9f
PUML: #54ff9f
SeaGreen2
HEX = #4eee94
PUML: #4eee94
SeaGreen3
HEX = #43cd80
PUML: #43cd80
SeaGreen
HEX = #2e8b57
PUML: SeaGreen
Aliases = sea green
SeaGreen4
PaleGreen1
HEX = #9aff9a
PUML: #9aff9a
LightGreen
HEX = #90ee90
PUML: LightGreen
Aliases = PaleGreen2
light green
PaleGreen3
HEX = #7ccd7c
PUML: #7ccd7c
PaleGreen4
HEX = #548b54
PUML: #548b54
SpringGreen
HEX = #00ff7f
PUML: SpringGreen
Aliases = spring green
SpringGreen1
SpringGreen2
HEX = #00ee76
PUML: #00ee76
SpringGreen3
HEX = #00cd66
PUML: #00cd66
SpringGreen4
HEX = #008b45
PUML: #008b45
green1
HEX = #00ff00
PUML: Lime
Aliases = green
green2
HEX = #00ee00
PUML: #00ee00
green3
HEX = #00cd00
PUML: #00cd00
green4
HEX = #008b00
PUML: #008b00
chartreuse1
HEX = #7fff00
PUML: Chartreuse
Aliases = chartreuse
chartreuse2
HEX = #76ee00
PUML: #76ee00
chartreuse3
HEX = #66cd00
PUML: #66cd00
chartreuse4
HEX = #458b00
PUML: #458b00
OliveDrab1
HEX = #c0ff3e
PUML: #c0ff3e
OliveDrab2
HEX = #b3ee3a
PUML: #b3ee3a
YellowGreen
HEX = #9acd32
PUML: YellowGreen
Aliases = yellow green
OliveDrab3
OliveDrab4
HEX = #698b22
PUML: #698b22
DarkOliveGreen1
HEX = #caff70
PUML: #caff70
DarkOliveGreen2
HEX = #bcee68
PUML: #bcee68
DarkOliveGreen3
HEX = #a2cd5a
PUML: #a2cd5a
DarkOliveGreen4
HEX = #6e8b3d
PUML: #6e8b3d


X11 rgb.txt - Gradients 5
khaki1
HEX = #fff68f
PUML: #fff68f
khaki2
HEX = #eee685
PUML: #eee685
khaki3
HEX = #cdc673
PUML: #cdc673
khaki4
HEX = #8b864e
PUML: #8b864e
LightGoldenrod1
HEX = #ffec8b
PUML: #ffec8b
LightGoldenrod2
HEX = #eedc82
PUML: #eedc82
LightGoldenrod3
HEX = #cdbe70
PUML: #cdbe70
LightGoldenrod4
HEX = #8b814c
PUML: #8b814c
LightYellow
HEX = #ffffe0
PUML: LightYellow
Aliases = light yellow
LightYellow1
LightYellow2
HEX = #eeeed1
PUML: #eeeed1
LightYellow3
HEX = #cdcdb4
PUML: #cdcdb4
LightYellow4
HEX = #8b8b7a
PUML: #8b8b7a
yellow1
HEX = #ffff00
PUML: Yellow
Aliases = yellow
yellow2
HEX = #eeee00
PUML: #eeee00
yellow3
HEX = #cdcd00
PUML: #cdcd00
yellow4
HEX = #8b8b00
PUML: #8b8b00
gold1
HEX = #ffd700
PUML: Gold
Aliases = gold
gold2
HEX = #eec900
PUML: #eec900
gold3
HEX = #cdad00
PUML: #cdad00
gold4
HEX = #8b7500
PUML: #8b7500
goldenrod1
HEX = #ffc125
PUML: #ffc125
goldenrod2
HEX = #eeb422
PUML: #eeb422
goldenrod3
HEX = #cd9b1d
PUML: #cd9b1d
goldenrod4
HEX = #8b6914
PUML: #8b6914
DarkGoldenrod1
HEX = #ffb90f
PUML: #ffb90f
DarkGoldenrod2
HEX = #eead0e
PUML: #eead0e
DarkGoldenrod3
HEX = #cd950c
PUML: #cd950c
DarkGoldenrod4
HEX = #8b6508
PUML: #8b6508
RosyBrown1
HEX = #ffc1c1
PUML: #ffc1c1
RosyBrown2
HEX = #eeb4b4
PUML: #eeb4b4
RosyBrown3
HEX = #cd9b9b
PUML: #cd9b9b
RosyBrown4
HEX = #8b6969
PUML: #8b6969
IndianRed1
HEX = #ff6a6a
PUML: #ff6a6a
IndianRed2
HEX = #ee6363
PUML: #ee6363
IndianRed3
HEX = #cd5555
PUML: #cd5555
IndianRed4
HEX = #8b3a3a
PUML: #8b3a3a
sienna1
HEX = #ff8247
PUML: #ff8247
sienna2
HEX = #ee7942
PUML: #ee7942
sienna3
HEX = #cd6839
PUML: #cd6839
sienna4
HEX = #8b4726
PUML: #8b4726

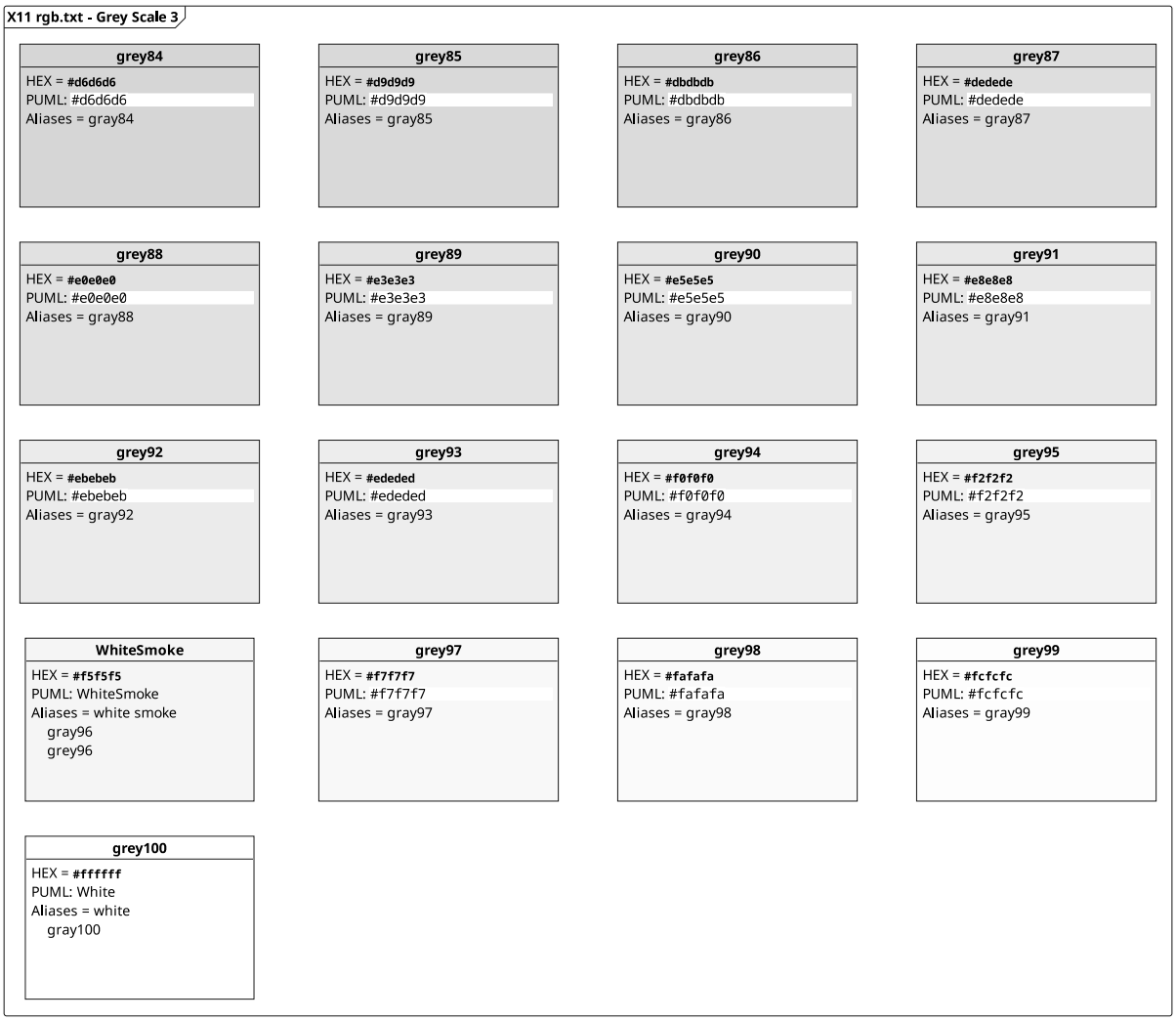

X11 rgb.txt - Gradients 6
burlywood1
HEX = #ffd39b
PUML: #ffd39b
burlywood2
HEX = #eec591
PUML: #eec591
burlywood3
HEX = #cdaa7d
PUML: #cdaa7d
burlywood4
HEX = #8b7355
PUML: #8b7355
wheat1
HEX = #ffe7ba
PUML: #ffe7ba
wheat2
HEX = #eed8ae
PUML: #eed8ae
wheat3
HEX = #cdba96
PUML: #cdba96
wheat4
HEX = #8b7e66
PUML: #8b7e66
tan1
HEX = #ffa54f
PUML: #ffa54f
tan2
HEX = #ee9a49
PUML: #ee9a49
tan3
HEX = #cd853f
PUML: Peru
Aliases = peru
tan4
HEX = #8b5a2b
PUML: #8b5a2b
chocolate1
HEX = #ff7f24
PUML: #ff7f24
chocolate2
HEX = #ee7621
PUML: #ee7621
chocolate3
HEX = #cd661d
PUML: #cd661d
SaddleBrown
HEX = #8b4513
PUML: SaddleBrown
Aliases = saddle brown
chocolate4
firebrick1
HEX = #ff3030
PUML: #ff3030
firebrick2
HEX = #ee2c2c
PUML: #ee2c2c
firebrick3
HEX = #cd2626
PUML: #cd2626
firebrick4
HEX = #8b1a1a
PUML: #8b1a1a
brown1
HEX = #ff4040
PUML: #ff4040
brown2
HEX = #ee3b3b
PUML: #ee3b3b
brown3
HEX = #cd3333
PUML: #cd3333
brown4
HEX = #8b2323
PUML: #8b2323
salmon1
HEX = #ff8c69
PUML: #ff8c69
salmon2
HEX = #ee8262
PUML: #ee8262
salmon3
HEX = #cd7054
PUML: #cd7054
salmon4
HEX = #8b4c39
PUML: #8b4c39
LightSalmon
HEX = #ffa07a
PUML: LightSalmon
Aliases = light salmon
LightSalmon1
LightSalmon2
HEX = #ee9572
PUML: #ee9572
LightSalmon3
HEX = #cd8162
PUML: #cd8162
LightSalmon4
HEX = #8b5742
PUML: #8b5742
orange1
HEX = #ffa500
PUML: Orange
Aliases = orange
orange2
HEX = #ee9a00
PUML: #ee9a00
orange3
HEX = #cd8500
PUML: #cd8500
orange4
HEX = #8b5a00
PUML: #8b5a00
DarkOrange1
HEX = #ff7f00
PUML: #ff7f00
DarkOrange2
HEX = #ee7600
PUML: #ee7600
DarkOrange3
HEX = #cd6600
PUML: #cd6600
DarkOrange4


X11 rgb.txt - Gradients 7
coral1
HEX = #ff7256
PUML: #ff7256
coral2
HEX = #ee6a50
PUML: #ee6a50
coral3
HEX = #cd5b45
PUML: #cd5b45
coral4
HEX = #8b3e2f
PUML: #8b3e2f
tomato1
HEX = #ff6347
PUML: Tomato
Aliases = tomato
tomato2
HEX = #ee5c42
PUML: #ee5c42
tomato3
HEX = #cd4f39
PUML: #cd4f39
tomato4
HEX = #8b3626
PUML: #8b3626
OrangeRed
HEX = #ff4500
PUML: OrangeRed
Aliases = orange red
OrangeRed1
OrangeRed2
HEX = #ee4000
PUML: #ee4000
OrangeRed3
HEX = #cd3700
PUML: #cd3700
OrangeRed4
HEX = #8b2500
PUML: #8b2500
red1
HEX = #ff0000
PUML: Red
Aliases = red
red2
HEX = #ee0000
PUML: #ee0000
red3
HEX = #cd0000
PUML: #cd0000
DarkRed
HEX = #8b0000
PUML: DarkRed
Aliases = red4
dark red
DebianRed
HEX = #d70751
PUML: #d70751
DeepPink
HEX = #ff1493
PUML: DeepPink
Aliases = deep pink
DeepPink1
DeepPink2
HEX = #ee1289
PUML: #ee1289
DeepPink3
HEX = #cd1076
PUML: #cd1076
DeepPink4
HEX = #8b0a50
PUML: #8b0a50
HotPink1
HEX = #ff6eb4
PUML: #ff6eb4
HotPink2
HEX = #ee6aa7
PUML: #ee6aa7
HotPink3
HEX = #cd6090
PUML: #cd6090
HotPink4
HEX = #8b3a62
PUML: #8b3a62
pink1
HEX = #ffb5c5
PUML: #ffb5c5
pink2
HEX = #eea9b8
PUML: #eea9b8
pink3
HEX = #cd919e
PUML: #cd919e
pink4
HEX = #8b636c
PUML: #8b636c
LightPink1
HEX = #ffaeb9
PUML: #ffaeb9
LightPink2
HEX = #eea2ad
PUML: #eea2ad
LightPink3
HEX = #cd8c95
PUML: #cd8c95
LightPink4
HEX = #8b5f65
PUML: #8b5f65
PaleVioletRed1
HEX = #ff82ab
PUML: #ff82ab
PaleVioletRed2
HEX = #ee799f
PUML: #ee799f
PaleVioletRed3
HEX = #cd6889
PUML: #cd6889
PaleVioletRed4
HEX = #8b475d
PUML: #8b475d


X11 rgb.txt - Gradients 8
maroon1
HEX = #ff34b3
PUML: #ff34b3
maroon2
HEX = #ee30a7
PUML: #ee30a7
maroon3
HEX = #cd2990
PUML: #cd2990
maroon4
HEX = #8b1c62
PUML: #8b1c62
VioletRed1
HEX = #ff3e96
PUML: #ff3e96
VioletRed2
HEX = #ee3a8c
PUML: #ee3a8c
VioletRed3
HEX = #cd3278
PUML: #cd3278
VioletRed4
HEX = #8b2252
PUML: #8b2252
magenta1
HEX = #ff00ff
PUML: Fuchsia
Aliases = magenta
magenta2
HEX = #ee00ee
PUML: #ee00ee
magenta3
HEX = #cd00cd
PUML: #cd00cd
DarkMagenta
HEX = #8b008b
PUML: DarkMagenta
Aliases = magenta4
dark magenta
orchid1
HEX = #ff83fa
PUML: #ff83fa
orchid2
HEX = #ee7ae9
PUML: #ee7ae9
orchid3
HEX = #cd69c9
PUML: #cd69c9
orchid4
HEX = #8b4789
PUML: #8b4789
plum1
HEX = #ffbbff
PUML: #ffbbff
plum2
HEX = #eeaeee
PUML: #eeaeee
plum3
HEX = #cd96cd
PUML: #cd96cd
plum4
HEX = #8b668b
PUML: #8b668b
MediumOrchid1
HEX = #e066ff
PUML: #e066ff
MediumOrchid2
HEX = #d15fee
PUML: #d15fee
MediumOrchid3
HEX = #b452cd
PUML: #b452cd
MediumOrchid4
HEX = #7a378b
PUML: #7a378b
DarkOrchid1
HEX = #bf3eff
PUML: #bf3eff
DarkOrchid2
HEX = #b23aee
PUML: #b23aee
DarkOrchid3
HEX = #9a32cd
PUML: #9a32cd
DarkOrchid4
HEX = #68228b
PUML: #68228b
purple1
HEX = #9b30ff
PUML: #9b30ff
purple2
HEX = #912cee
PUML: #912cee
purple3
HEX = #7d26cd
PUML: #7d26cd
purple4
HEX = #551a8b
PUML: #551a8b
MediumPurple1
HEX = #ab82ff
PUML: #ab82ff
MediumPurple2
HEX = #9f79ee
PUML: #9f79ee
MediumPurple3
HEX = #8968cd
PUML: #8968cd
MediumPurple4
HEX = #5d478b
PUML: #5d478b
thistle1
HEX = #ffe1ff
PUML: #ffe1ff
thistle2
HEX = #eed2ee
PUML: #eed2ee
thistle3
HEX = #cdb5cd
PUML: #cdb5cd
thistle4

20.4 X11 Colors - Grey Scale


X11 rgb.txt - Grey Scale 2
grey40
HEX = #666666
PUML: #666666
Aliases = gray40
DimGray
HEX = #696969
PUML: DimGray
Aliases = dim gray
dim grey
DimGrey
gray41
grey41
grey42
HEX = #6b6b6b
PUML: #6b6b6b
Aliases = gray42
grey43
HEX = #6e6e6e
PUML: #6e6e6e
Aliases = gray43
grey44
HEX = #707070
PUML: #707070
Aliases = gray44
grey45
HEX = #737373
PUML: #737373
Aliases = gray45
grey46
HEX = #757575
PUML: #757575
Aliases = gray46
grey47
HEX = #787878
PUML: #787878
Aliases = gray47
grey48
HEX = #7a7a7a
PUML: #7a7a7a
Aliases = gray48
grey49
HEX = #7d7d7d
PUML: #7d7d7d
Aliases = gray49
grey50
HEX = #7f7f7f
PUML: #7f7f7f
Aliases = gray50
grey51
HEX = #828282
PUML: #828282
Aliases = gray51
grey52
HEX = #858585
PUML: #858585
Aliases = gray52
grey53
HEX = #878787
PUML: #878787
Aliases = gray53
grey54
HEX = #8a8a8a
PUML: #8a8a8a
Aliases = gray54
grey55
HEX = #8c8c8c
PUML: #8c8c8c
Aliases = gray55
grey56
HEX = #8f8f8f
PUML: #8f8f8f
Aliases = gray56
grey57
HEX = #919191
PUML: #919191
Aliases = gray57
grey58
HEX = #949494
PUML: #949494
Aliases = gray58
grey59
HEX = #969696
PUML: #969696
Aliases = gray59
grey60
HEX = #999999
PUML: #999999
Aliases = gray60
grey61
HEX = #9c9c9c
PUML: #9c9c9c
Aliases = gray61
grey62
HEX = #9e9e9e
PUML: #9e9e9e
Aliases = gray62
grey63
HEX = #a1a1a1
PUML: #a1a1a1
Aliases = gray63
grey64
HEX = #a3a3a3
PUML: #a3a3a3
Aliases = gray64
grey65
HEX = #a6a6a6
PUML: #a6a6a6
Aliases = gray65
grey66
HEX = #a8a8a8
PUML: #a8a8a8
Aliases = gray66
grey67
HEX = #ababab
PUML: #ababab
Aliases = gray67
grey68
HEX = #adadad
PUML: #adadad
Aliases = gray68
grey69
HEX = #b0b0b0
PUML: #b0b0b0
Aliases = gray69
grey70
HEX = #b3b3b3
PUML: #b3b3b3
Aliases = gray70
grey71
HEX = #b5b5b5
PUML: #b5b5b5
Aliases = gray71
grey72
HEX = #b8b8b8
PUML: #b8b8b8
Aliases = gray72
grey73
HEX = #bababa
PUML: #bababa
Aliases = gray73
grey74
HEX = #bdbdbd
PUML: #bdbdbd
Aliases = gray74
grey75
HEX = #bfbfbf
PUML: #bfbfbf
Aliases = gray75
grey76
HEX = #c2c2c2
PUML: #c2c2c2
Aliases = gray76
grey77
HEX = #c4c4c4
PUML: #c4c4c4
Aliases = gray77
grey78
HEX = #c7c7c7
PUML: #c7c7c7
Aliases = gray78
grey79
HEX = #c9c9c9
PUML: #c9c9c9
Aliases = gray79


X11 rgb.txt - Grey Scale 3
grey84
HEX = #d6d6d6
PUML: #d6d6d6
Aliases = gray84
grey85
HEX = #d9d9d9
PUML: #d9d9d9
Aliases = gray85
grey86
HEX = #dbdbdb
PUML: #dbdbdb
Aliases = gray86
grey87
HEX = #dedede
PUML: #dedede
Aliases = gray87
grey88
HEX = #e0e0e0
PUML: #e0e0e0
Aliases = gray88
grey89
HEX = #e3e3e3
PUML: #e3e3e3
Aliases = gray89
grey90
HEX = #e5e5e5
PUML: #e5e5e5
Aliases = gray90
grey91
HEX = #e8e8e8
PUML: #e8e8e8
Aliases = gray91
grey92
HEX = #ebebeb
PUML: #ebebeb
Aliases = gray92
grey93
HEX = #ededed
PUML: #ededed
Aliases = gray93
grey94
HEX = #f0f0f0
PUML: #f0f0f0
Aliases = gray94
grey95
HEX = #f2f2f2
PUML: #f2f2f2
Aliases = gray95
WhiteSmoke
HEX = #f5f5f5
PUML: WhiteSmoke
Aliases = white smoke
gray96
grey96
grey97
HEX = #f7f7f7
PUML: #f7f7f7
Aliases = gray97
grey98
HEX = #fafafa
PUML: #fafafa
Aliases = gray98
grey99
HEX = #fcfcfc
PUML: #fcfcfc
Aliases = gray99
grey100
HEX = #ffffff
PUML: White
Aliases = white
gray100

CHAPTER

TWENTYONE

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.

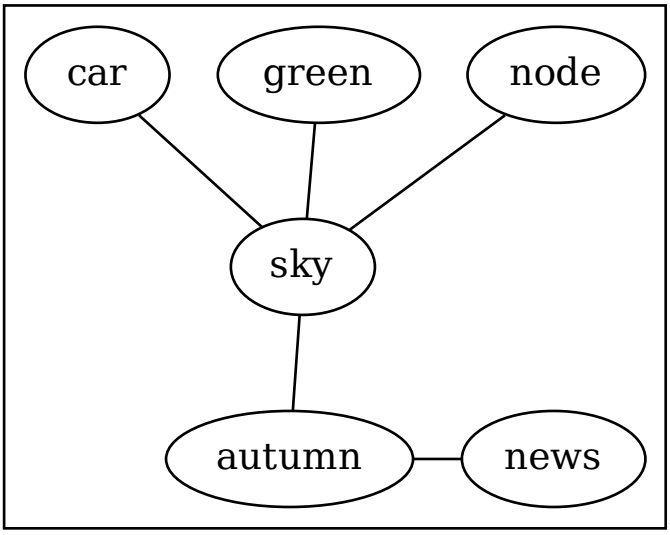
21.1 Trees

21.1.1 Definition of a Tree

The most useful definition of a tree, comes from graph theory, where

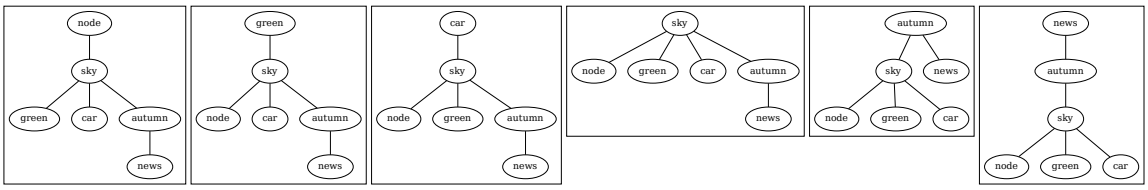
> a tree is an undirected graph in which any two vertices are connected by exactly one path. Every acyclic connected graph is a tree, and vice versa. A forest is a disjoint union of trees, or equivalently an acyclic graph that is not necessarily connected.

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 distcinction 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:

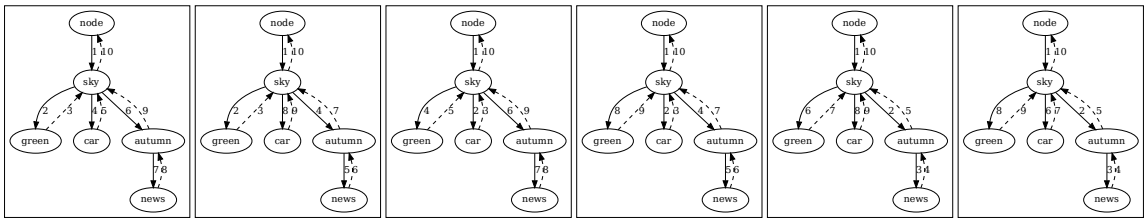
- depth-first search and
- breadth-first search.

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

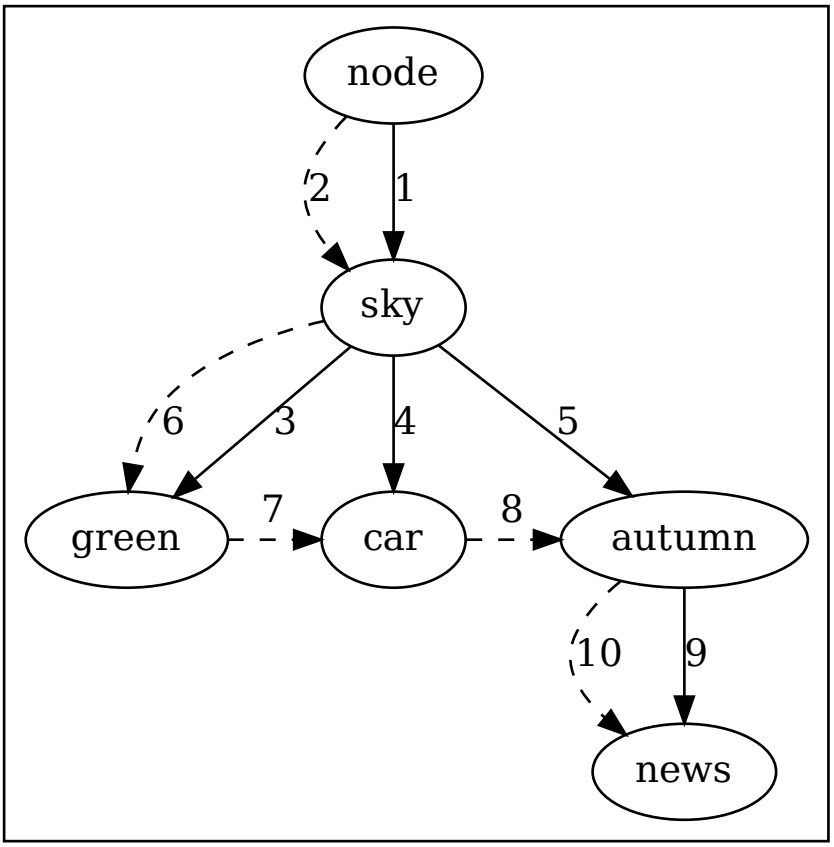
The most common way to walk around a tree is a depth-first search. This tree traversal is also presented with an implicit order of nodes, which is unnecessary and therefore belongs to the set of *unfounded brain-dead assumptions*.

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:



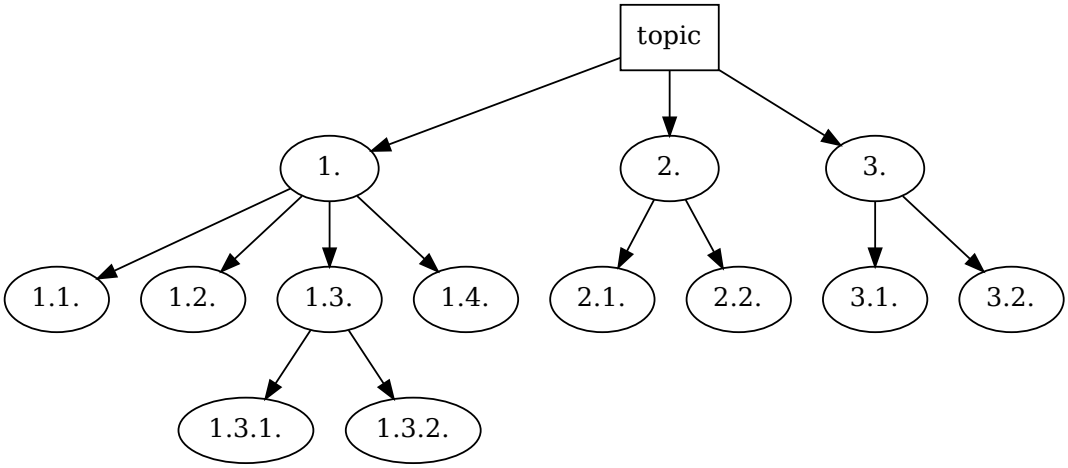
Illustrative Video

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 ONeillCode 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
>
> ONeill 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

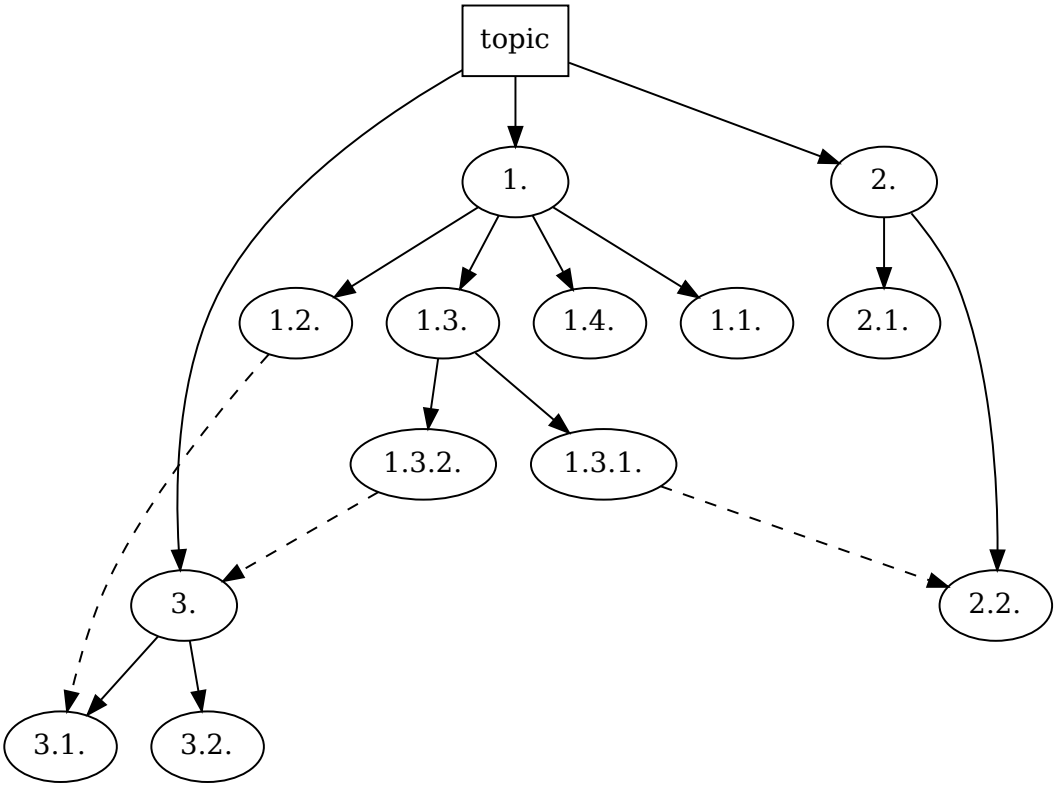
The organization of knowledge in books is equivalent to an ordered depth-first search.



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.

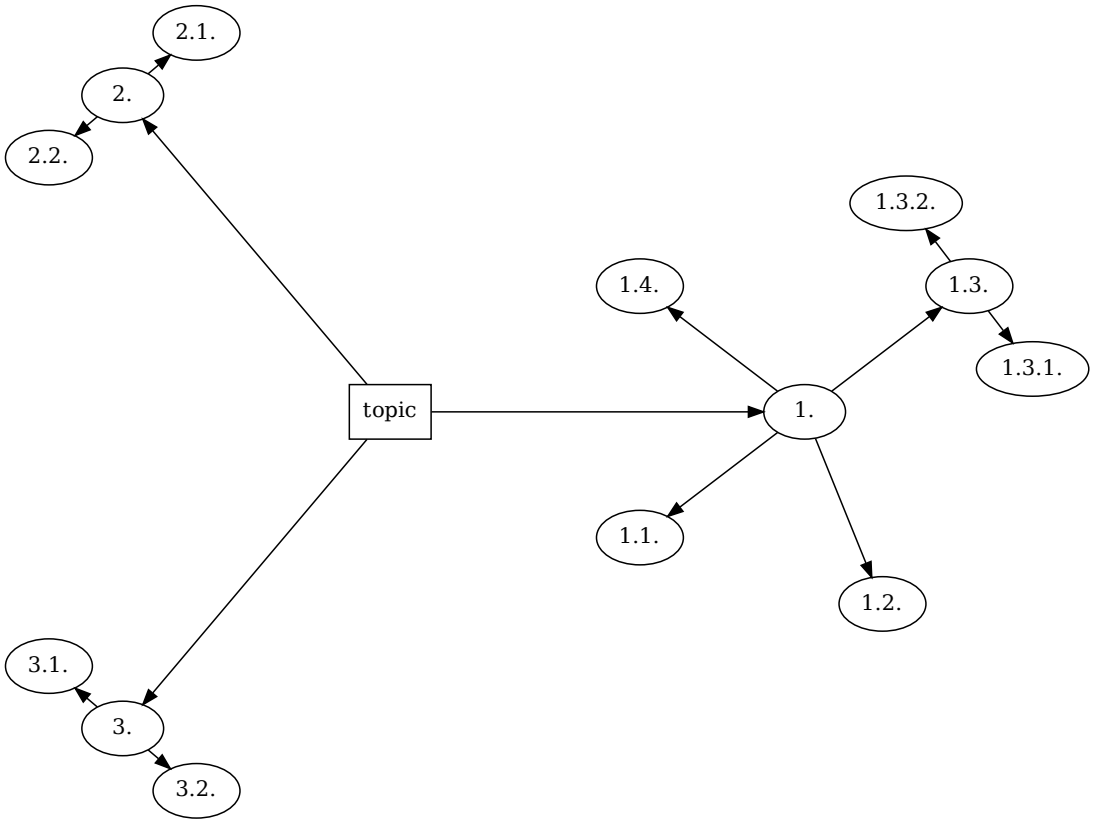
Additional pathes, which violate the tree conditions can be introduced with cross-references:



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:

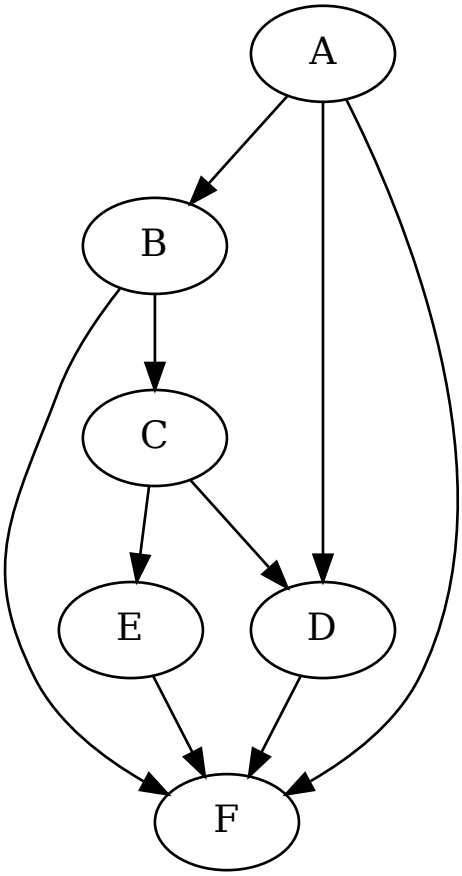


21.2 Generalization

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;
```

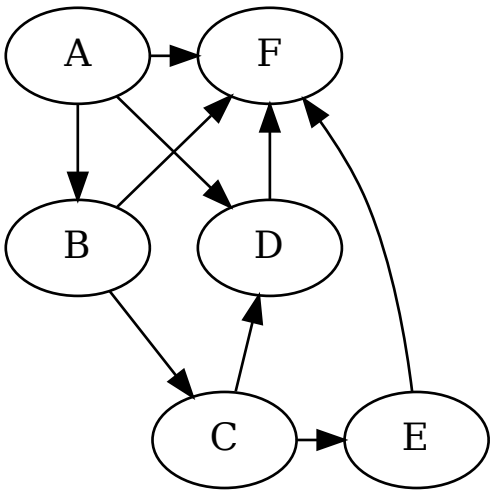
graphviz dot(1) draws an acceptable graph, which is quite tree like:



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; }
```

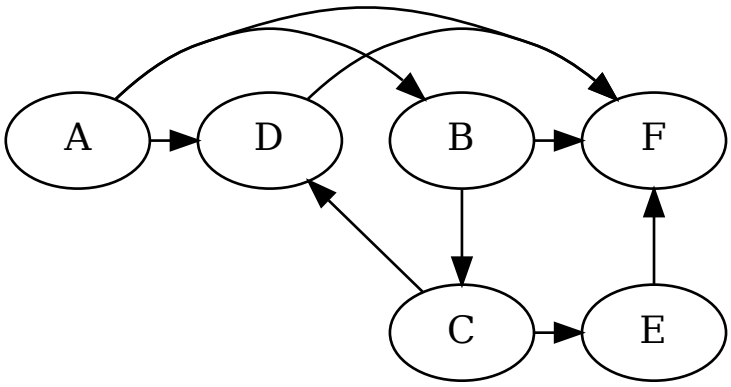
the graph looks less like a tree and more like a graph:



With one additonal restriction:

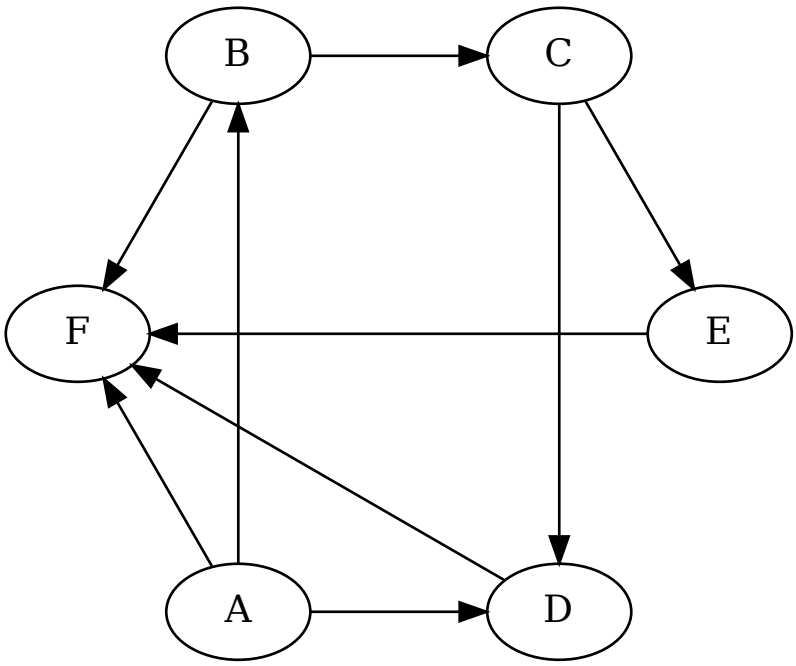
```
subgraph { rank = same; A; F; }
subgraph { rank = same; B; D; }
subgraph { rank = same; C; E; }
subgraph { rank = same; B; F; }
```

the faint resemblance of a tree is gone:

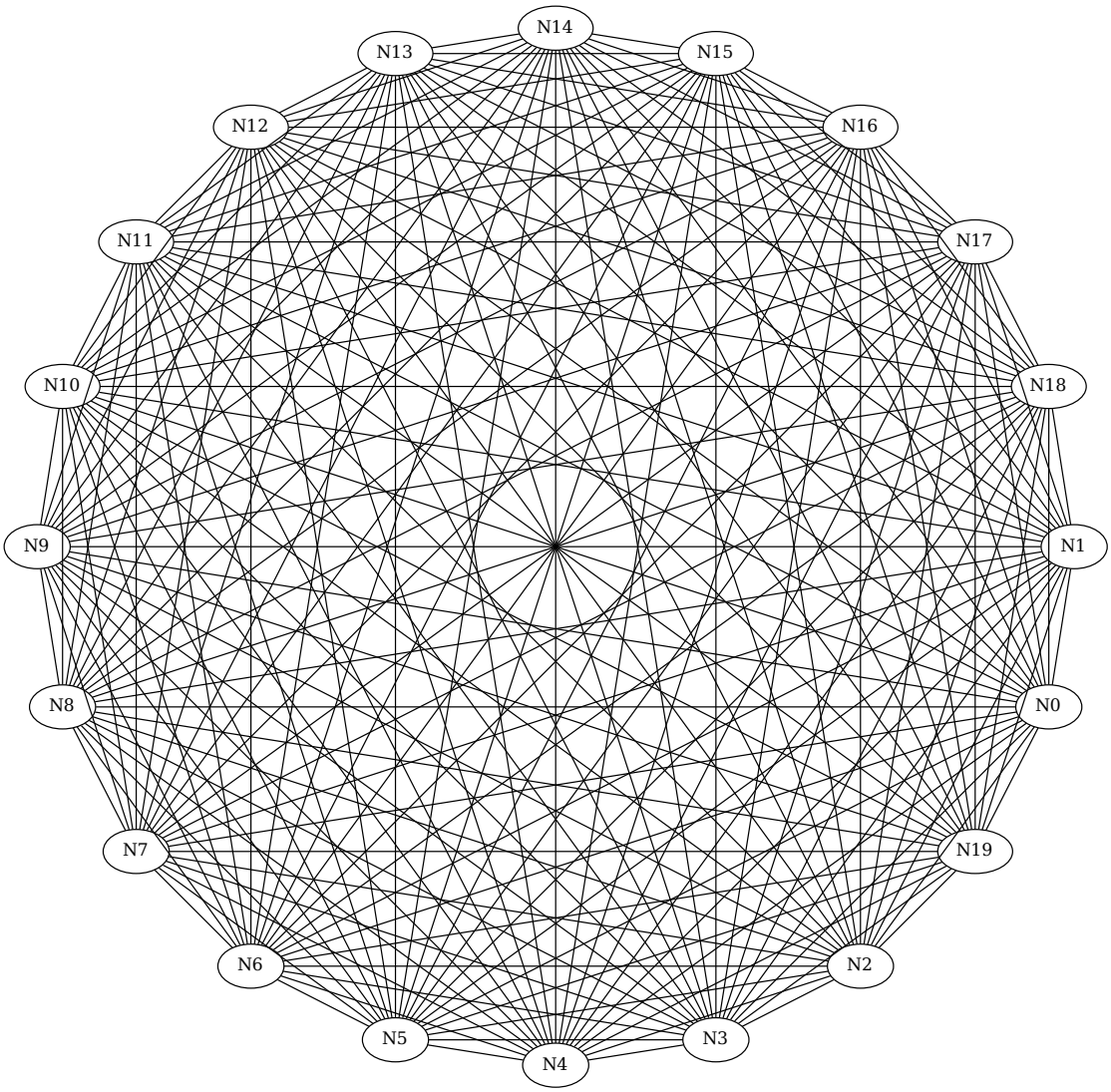


21.2.1 Circular Graphs

Giving up on trees, the most obvious layout to accomodate 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:

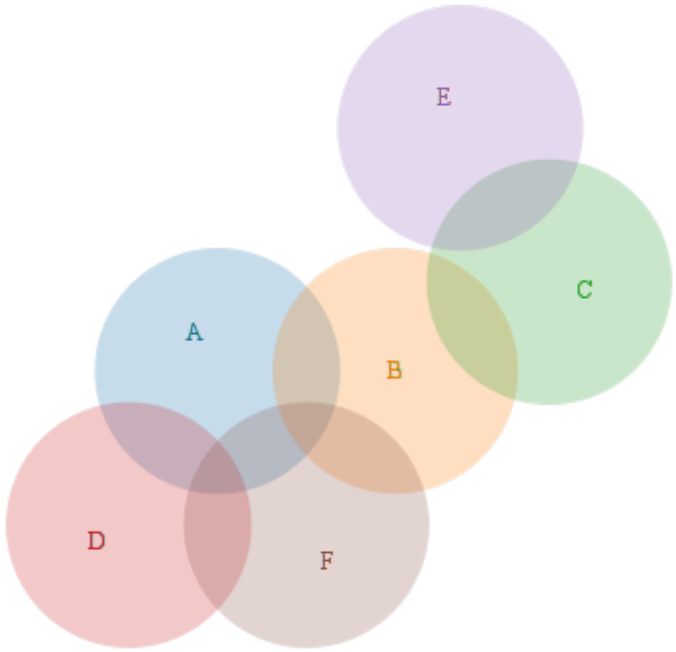
```
<script src="https://d3js.org/d3.v4.min.js"></script>
<script src="http://benfred.github.io/venn.js/examples/../venn.js"></script>
<script>
// define sets and set set intersections
var sets = [
      {sets: ['A'], size: 12},
      {sets: ['B'], size: 12},
      {sets: ['C'], size: 12},
      {sets: ['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 assumptions 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 possiblities 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.

CHAPTER

TWENTYTWO

SCRATCH

Miscellaneous stuff to be ordered and processed later.

22.1 Style Guide

Despite the cpaitalization 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 capitalizaton 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 *here* tag
M-h	go to next *here* tag. If not found, go to previous *here* tag
C-u M-<up>	define *here* tag and go to previous *here* 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 here 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

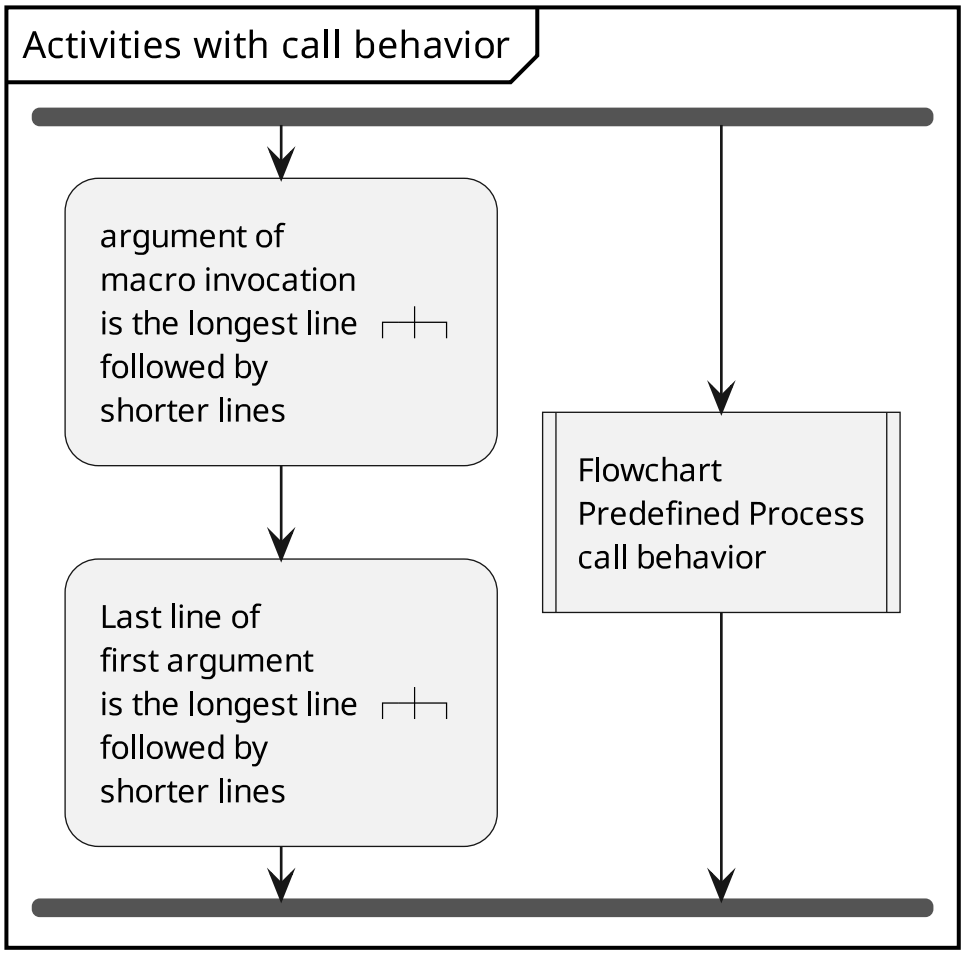
PlantUML has no built-in support for the rake symbol (⏚) which designates call behavior in UML 2.x. _static/call-behavior.puml contains the macros RAKE and CALL, which use UNICODE block characters to provide it.

```
/' |: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:

```
30 matches for "\(\(\(\(^\([  ^I^M]*\(def\|class\)[ ]+\|[A-Za-z_]?\)\|\`\)\)\([-A-Za-z_][-0-9A-Za-z_]*\)\)\)\([     ^I^M]*(\
     75:================================================
     76:# |||:sec:|||  COMPATIBILITY
     77:================================================
     90:         def printf(*args, **kwd):  # 2.4, 2.5, define our own Print function
    105:     def sformat  (fmtspec, *args, **kwargs):
    110:         def sformat  (fmtspec, *args, **kwargs):
    122:     def isstring( obj):
    129:         def isstring(obj):
    133:     def issequence(arg, or_dict= False, or_seq= True):            # ||:fnc:||
    148:     def sequence_type(value):                                     # ||:fnc:||
    160:def uchar(num):
    204:def lfind(l, elt):
    213:================================================
    214:# |||:sec:|||  CONFIGURATION
    215:================================================
    217:__all__ = []
    218:__all_internal__  = []
    221:dbg_fwid  = globals().get('dbg_fwid', 15)
    251:================================================
    252:# |||:sec:|||  CLASSES
    253:================================================
    257:================================================
    258:# |||:sec:|||  FUNCTIONS
    259:================================================
    269:================================================
    270:# |||:sec:|||  UTILITIES
    271:================================================
    284:def printe_(*args, **kwargs):
    302:__all__ = list(reversed(__all__))
    304:def run(parameters):                                             # ||:fnc:||
    316:================================================
    317:# |||:sec:|||  MAIN
    318:================================================
    320:_quiet = False
    321:_verbose = False
    322:_debug = False
    325:file_encoding_is_clean  = False
    326:def setdefaultencoding (encoding= None, quiet= False):
    355:def main(argv):                                                  # ||:fnc:||
    383:     class AdHocAction(argparse.Action):
    384:         def __call__(self, parser, namespace, values, option_string=  None):
    427:         def args_split_range(args):
```

22.6 Tastatur- und Spracheinstellung

Eventuell nach dem Start, Tastaturbelegung einstellen.

Ohne Einstellung der Tastaturbelegung befinden sich folgende Tasten an anderer Stelle:

Beschriftung	Zeichen
z	y
y	z
ß	-
-	/

22.7 Build statistics

Add new statistic data to `doc/_static/build-stats.csv`:

```
( cd "${HOME}"/project/documentation/bin && ./build-stats-for-hosts.sh --no-header >> ../doc/_
↪static/build-stats.csv ws@sheckley )
```

host	processors	bogomips	memory	real	user	sys	load1	load5	load15
luna.wiedenmann.intern	1	7182.70	10G	13m33.565s	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	4	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.

CHAPTER

TWENTYTHREE

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.

   - substitue 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 excatly the same as an enum in C/C++:

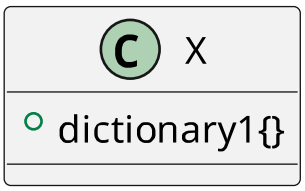
```
enum { xyz = 2, abc, def=5, ghi };

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 evey 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?

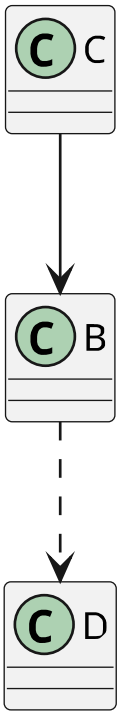
9. Do I need more than E, C and G for describing every thinkable programm?

   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, Gerneralization/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.

CHAPTER

TWENTYFOUR

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 expresssion, `RET`),

```
+-------+-------+-------+
|       |       |   9   |
|       |   8   |       |
|   7   _____________________ • /home/sw/project/documentation/_static/rst-TAGS
+-------+-------+-------+
|       |       |   6   |
|       |   5   |       |
|   4   |       |       |
+-------+-------+-------+
|       |       |   3 _____ • /home/ws/project/documentation/bin/diffmap.py
|       |   2   _____________ • /home/ws/project/documentation/bin/rst_tags.py
|   1   _____________________ • /home/sw/project/documentation/README-rst-mode-etags-support.txt
+-------+-------+-------+
|       |
|   0   |
|       |
+-------+
```

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)

```
ws-admin/TAGS,include
^L
ws-admin/tagideasy/TAGS,include
^L
ws-bme/TAGS,include
```

This file should be installed in directory "$*HOME*/project" as TAGS with:

```
test -n "$( dpkg -l coreutils 2>/dev/null | grep ^ii )" || sudo apt-get install coreutils
printf
↪'DApkb2N1bWVudGF0aW9uL1RBR1MsaW5jbHVkZQoMCmFkbWluaXN0cmF0aW9uL1RBR1MsaW5jbHVkZQoMCndzLWFkbWluL1RBR1MsaW5jbHV
↪' | base64 -d >TAGS
```

M-x t-r-t-t expands to tags-reset-tags-tables() which allows changing the active TAGS file.

The Emacs rst tags support is supplied by find-tag-default-rst(), which is already integrated in package emacs-init.

Etags generation is provided by **rst_tags.py**, which is also integrated in the snippets framework:

```
sda make tags
```

For this to work, a project update (sda update) might be needed, or the *tags* rule can be extracted from the snippet with:

```
snc -m mak '^mak_sphinx.makefile-footer$' | sed '/^tags:/p;1,/tags:/d' | sed '/^ *$/,$d'
```

which at the time of this rendering was:

```

```

24.2.1 TAGS File Structure

A TAGS file has one or more entries describing the tags in a file, see e.g. Understanding the 'ctags -e' file format (ctags for emacs) - Stack Overflow.

An etags file entry consists of

```
<\014> file name "," size of following tag entries
```

An etags tag entry has the following structure:

```
line text <\177> tag <\001> line no (1-based) "," char position (0-based)
```

24.2.2 Sample TAGS File

The TAGS file as seen in Emacs:

```
^L
README-rst-mode-etags-support.txt,57
.. _`sec:Sample TAGS File`:^?sec:Sample TAGS File^A21,6337
^L
../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)))
```

```
>>> 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])
```

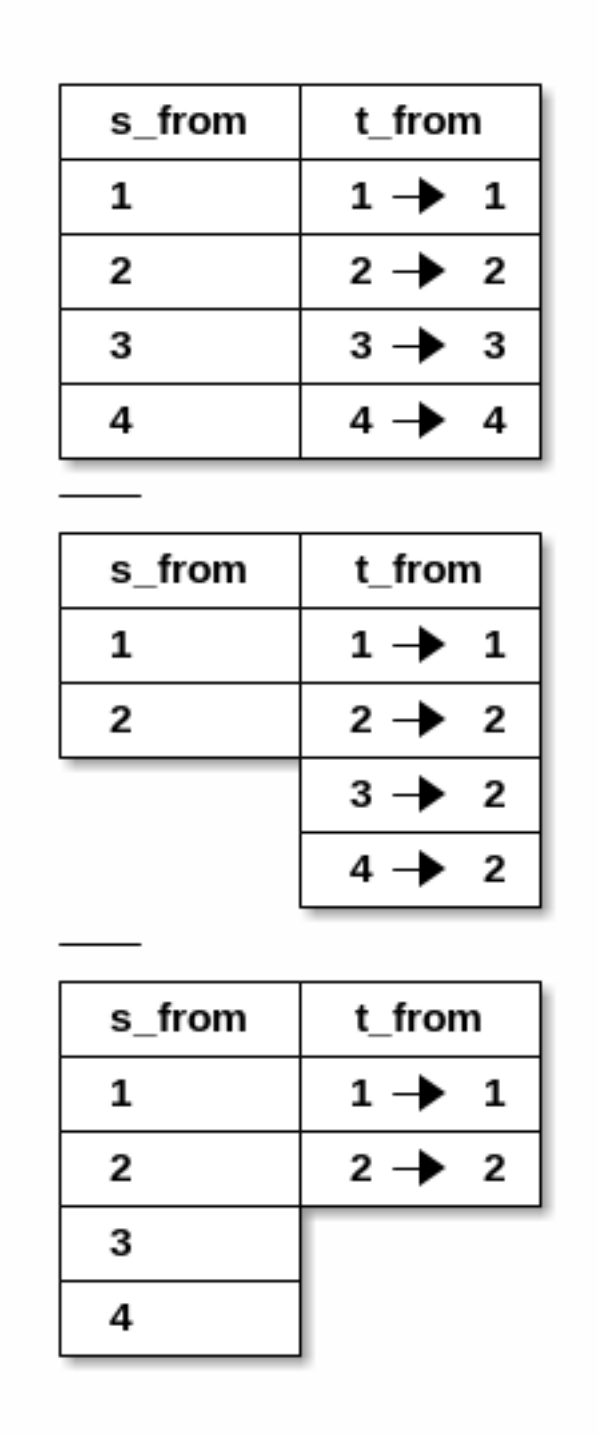


figure 24.1: Correlate Line Numbers from Diff Output

```
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   <---/                 \  These are between line 03
05 | line 5   <--/                   > and line 04 of target file
06 | line 6   <-/                   /
07 | line 7   <-----   04 | line 7
08 | line 8   <-----   05 | line 8
```

```
--- 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
```

(continues on next page)

(continued from previous page)

```
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*)

```
>>> _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)
```

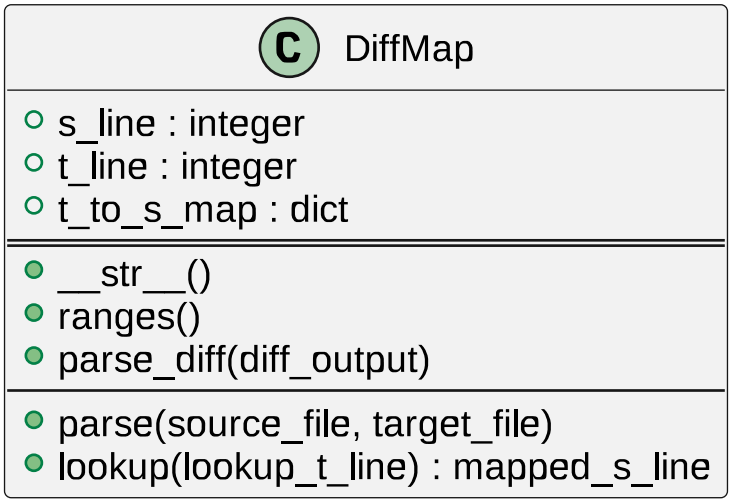


figure 24.2: Correlate Line Numbers from Diff Output

```
>>> 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
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
```

(continues on next page)

(continued from previous page)

```
+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

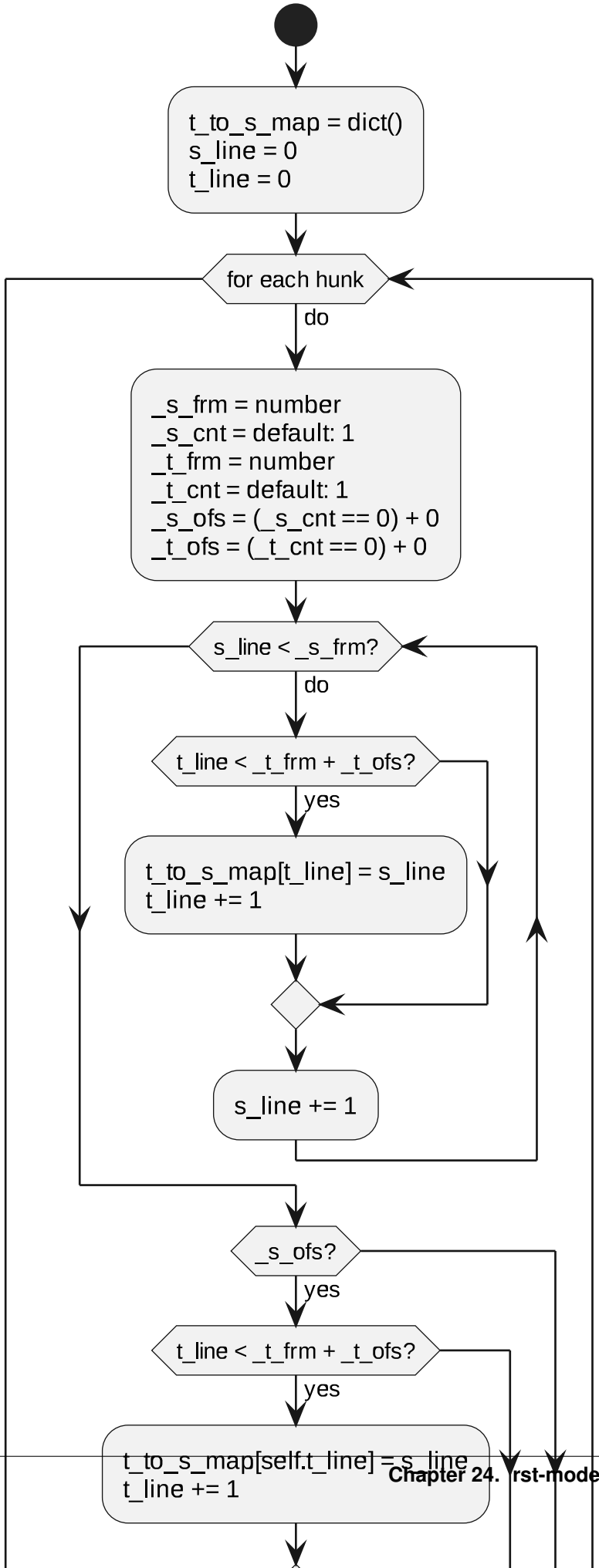
```
printf "%s\n%s\n%s\n" 'hallo' 'hallo' 'hallo' | diff -u /dev/null -
```

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


t_to_s_map = dict()
s_line = 0
t_line = 0
for each hunk
do
_s_frm = number
_s_cnt = default: 1
_t_frm = number
_t_cnt = default: 1
_s_ofs = (_s_cnt == 0) + 0
_t_ofs = (_t_cnt == 0) + 0
s_line < _s_frm?
do
t_line < _t_frm + _t_ofs?
yes
t_to_s_map[t_line] = s_line
t_line += 1
s_line += 1
_s_ofs?
yes
t_line < _t_frm + _t_ofs?
yes
t_to_s_map[self.t_line] = s_line
t_line += 1

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

INDEX

R

S

X