

Deplate 0.8.3 – convert wiki-like markup to latex, docbook, html, or “html-slides”

Thomas Link

10. Jul 2008

Keywords: Converter, Text, LaTeX, HTML, Docbook, Wiki, Presentation, Web Publishing

Contents

1	Introduction	7
1.1	Goals	8
2	Getting deplate	8
2.1	Requirements	8
2.2	Download	9
2.3	Support	9
3	Installation	9
3.1	Gem	9
3.2	Win32	9
3.3	ZIP Archive	10
3.4	Related Software	11
4	Usage	11
4.1	Editor support	15
4.1.1	General remark	15
4.1.2	VIM	16
5	Configuration	16
5.1	User configuration of the standard setup	17
5.2	Configuration of wiki names	17
5.3	Configuration via deplate.ini	18
5.4	Options	19
5.5	Templates	20
5.5.1	Template variables (obsolete)	20
5.5.2	Template files	21

6	Input Formats	22
6.1	deplate	23
6.2	deplate-restricted	23
6.3	deplate-headings	23
6.4	rdoc	23
6.4.1	heading (rdoc, level 3)	24
6.5	play – Hypothetical support for screen-plays and stage-plays	25
6.5.1	EXT. Beach – Day	27
6.6	template	27
7	Output Formats	27
7.1	HTML, single-file (no chunks) output	27
7.2	HTML Site, multi-page (chunky) output	28
7.3	HTML Slides, abbreviated online presentations	28
7.4	HTML Website	28
7.5	XHTML 1.0-transitional (xhtml10t)	28
7.6	XHTML 1.1 with MathML (xhtml11m)	28
7.7	Php, PhpSite	28
7.8	LaTeX	29
7.8.1	latex-dramatist: Typeset stage plays with the dramatist package	29
7.9	Docbook: article, book, reference pages	30
7.10	Plain text	30
7.11	Template	30
8	Modules	31
8.1	Localization: lang-en, lang-de, lang-ru-koi8-r, lang-zh_CN	31
8.1.1	guesslanguage: Guess the locale	31
8.2	General	32
8.2.1	anyword: Turn specified words into hyperlinks	32
8.2.2	code-coderay: Highlight #Code regions using coderay	32
8.2.3	code-gvim, code-gvim71: Highlight #Code regions using gvim	33
8.2.4	code-highlight: Highlight #Code regions using gvim	33
8.2.5	endnotes: Turn footnotes into endnotes	33
8.2.6	entities-decode: Insert characters as entities	33
8.2.7	entities-encode: Insert characters as entities	33
8.2.8	iconv: Convert text between encodings	34
8.2.9	imgurl: Insert urls/links to images as images	34
8.2.10	linkmap: Define labels refering to URLs	34
8.2.11	makefile: Create Makefiles	35
8.2.12	mark-external-urls: Mark external references with icons	35
8.2.13	numpara: Numbered paragraphs	36
8.2.14	particle-math: LaTeX-like \math markup	36
8.2.15	smart-dash: Insert long or short dashes depending on the context	36
8.2.16	smiley: Replace text smileys with images	36
8.2.17	utf8: Improve unicode awareness	37
8.2.18	xmlrpc: Work as a xmlrpc server	37
8.3	Syntax	38
8.3.1	syntax-region-alt: Alternative syntax for regions	38

8.3.2	markup-1: Re-enable pre0.6 text styles	38
8.4	LaTeX	38
8.4.1	inlatex-compound: Compile LaTeX bits as one file	38
8.4.2	koma: Support for the Koma-Script class	39
8.4.3	latex-emph-table-head: Emphasize head rows	39
8.4.4	latex-verbatim-small: Print verbatim regions in small font size	39
8.4.5	latex-styles: Styled LaTeX output	39
8.5	HTML	39
8.5.1	soffice: Support for Star/Sun/OpenOffice HTML	39
8.5.2	html-asciimath: Support for ASCIIMathML.js	39
8.5.3	html-headings-navbar: Insert a navigation bar before level 1-headings	39
8.5.4	html-jsmath: Support for jsMath.js	40
8.6	html-mathml: Support for the mathml gem	40
8.6.1	html-obfuscate-email: Obfuscate e-mail	40
8.6.2	html-sidebar: Display a sidebar containing a small table of contents	40
8.6.3	navbar-png: Display images in the navigation bar	40
8.7	HTML Slides	40
8.7.1	htmlslides-navbar-fh: Modified navigation bar	40
8.8	Docbook	40
8.8.1	symbols-*: Modify the representation of symbols	40
8.8.2	noindent: Avoid insertion of spaces and newlines	40
8.9	Miscellaneous	41
8.9.1	DeplateString	41
8.9.2	Nukumi2	41
9	Markup	41
9.1	The Basics	41
9.1.1	Elements and Particles	41
9.1.2	Backslashes	42
9.1.3	Special Characters	42
9.1.4	Argument Values	43
9.2	Comments (whole lines)	44
9.3	Paragraphs	44
9.4	Headings	45
9.5	Lists (indented)	45
9.6	Description lists (indented)	46
9.7	Tables	47
9.8	Anchors	51
9.9	Wiki Names, URLs	51
9.10	Symbols	52
9.11	Markers	53
9.12	Notes	53
9.13	Strings, Smart Quotes	53
9.14	Textstyles	54
9.15	Breaks	54
9.16	Whitespace	54

10 Regions	54
10.1 Abstract	55
10.2 Code	56
10.3 Define: DefRegion, DefCommand, DefMacro, DefElement, DefRegionN, DefCommandN, DefMacroN	56
10.3.1 Elements	59
10.3.2 Particles	59
10.3.3 Native templates	59
10.4 Doc, Var	60
10.5 For	60
10.6 Header, Footer	61
10.7 Img	61
10.8 Inlatex, Ltx	62
10.9 Native	65
10.10Quote	65
10.11Set, Get	66
10.12R generated tables	66
10.13Region	68
10.14Ruby	68
10.15Swallow, Skip	69
10.16Table	69
10.17Verbatim	69
10.18Write	70
11 Commands	70
11.1 Getting or setting data about the document	70
11.2 Flow control, document management	72
11.3 Bibliography	74
11.4 Abbreviations, index	74
11.5 Dynamic text, elements	75
12 Macros	77
12.1 Getting or setting data about the document	78
12.2 References, labels, index	79
12.3 Bibliography	80
12.4 Textstyles	80
12.5 Dynamic text, particles	80
13 Skeletons	83
14 Variables and options	85
14.1 Document variables	85
14.1.1 Pre-defined variables	85
14.1.2 General	85
14.1.3 Output Format	89
14.1.4 Module	91
14.1.5 Legacy	93
14.2 Strings, booleans, arrays, and hashes	93
14.3 Template “services”	95

14.4	Element properties	96
14.4.1	Global properties	96
14.5	Tags and filters	96
14.5.1	Foo	97
14.6	Special arguments	97
14.7	Clips	98
15	Internals	98
15.1	Document structure	98
16	Extending deplate	100
16.1	Line/Elements	100
16.2	Text/Particles	100
16.2.1	Symbols	100
16.2.2	Macros	101
16.3	Formatters	101

List of Tables

2	How the zh_CN modules set spaces	32
3	Key arguments	44
4	This Table	48
5	A clipped table	66
6	An example from the xtable manual using the tli data set	67
7	Summary of the tli data set	67
8	Summary of the tli data set (guessing cell borders)	68
9	An example of an char separated table	69

List of Figures

1	Example based on the dot manual	63
2	Example plot based on Venables/Ripley (2003)	63
3	An example from the xypic-user guide	65
4	A nice drawing	77

List of Examples

4.1	Using the Deplate::Converter class
5.1	deplate.ini
5.2	A letter template
6.1	RDoc input
6.2	Play input
7.1	Php output
8.1	Entities
8.2	The makefile module
8.3	The numpara module: Don't number specific buffers
8.4	The numpara module: Change numbering style

- 8.5 Invoke as a xmlrpc server
- 8.6 The syntax-region-alt module
- 9.1 Text styles, backslash
- 9.2 Comments
- 9.3 Paragraphs
- 9.4 List
- 9.5 Description list
- 9.6 Table
- 9.7 Table styles
- 9.8 Anchors
- 9.9 Wiki names
- 9.10 Symbols
- 9.11 Markers
- 9.12 Notes
- 9.13 Quotes
- 9.14 Text styles
- 10.1 Abstract
- 10.2 Code, syntax highlighting
- 10.3 Define a new region
- 10.4 Example region
- 10.5 Example region
- 10.6 Literate programming
- 10.7 Define an element
- 10.8 Define a new particle
- 10.9 Insert text literally
- 10.10 For
- 10.11 Image created with dot
- 10.12 Inline LaTeX
- 10.13 “Native” text
- 10.14 Quotations
- 10.15 Clip
- 10.16 Tables created with R
- 10.17 Inline ruby
- 10.18 Create a table from tab-separated input
- 10.19 Verbatim
- 10.20 Write something to a file
- 11.1 Commands
- 12.1 Macro written in Ruby
- 12.2 Macros
- 13.1 Skeletons (1)
- 13.2 Skeletons (2)
- 14.1 Defining variables
- 14.2 Variable types
- 14.3 Templates
- 14.4 Global properties
- 14.5 Tags and filters

This manual was created with deplate version 0.8.3.

1 Introduction

`deplate` is a ruby based tool for converting documents written in wiki-like markup to LaTeX, HTML, “HTML slides”, or docbook. It supports page templates, embedded LaTeX code, footnotes, citations, bibliographies, automatic generation of an index, table of contents etc. It can be used to create web pages and (via LaTeX or Docbook) high-quality printouts from the same source.

Although originally created as a LaTeX (or word-processor) replacement for “everyday use”, `deplate` probably isn’t suited for highly technical documents or documents that require sophisticated graphical layout. For other purposes it should work fine. As the html output filter provides menus, templates, indexes, and the like, within some limits, it could also qualify as offline CMS for maintaining small websites. If you want to do something bigger you might want to use a different program though.

`deplate` *aims* to be modular and easily extensible. It is the accompanying converter for the Vim wiki plugin. In the family of wiki engines, the “native” `deplate` markup originated from the emacs-wiki.

`deplate` reads:

- `deplate` markup (wiki vim plugin) in several variants: full, restricted, template (this markup is explained in Markup (p. 41) and the following chapters)
- ruby’s rdoc

`deplate` writes:

- HTML: single page, web site (chunked), web-based presentation (chunked & condensed), as PHP page
- LaTeX (which can be converted to dvi, postscript, pdf ...)
- Docbook (which can be converted to html, rtf, pdf, xml-fo, man ...)
- Really plain text

A note on code quality: This is a personal/private pet project that started off as a miniscule script to quickly convert emacs-wiki pages without emacs. This program is the result of a long series of early morning pre-coffee plumbing and patch-work. It works well for me tough and has proven quite useful.

In case `deplate` doesn’t meet your needs, there are plenty of similar tools you might want to give a try.

For converting mostly plain text to HTML, LaTeX etc.:

- aft
- PlainDoc (pd2tex)
- reStructuredText
- sisu
- txt2tags
- yodl

With respect to website generation, you might also be interested in these tools:

- webgen
- rote

The authors of `zoem` (Stijn van Dongen and Joost van Baal) compiled a comprehensive list of similar efforts.

1.1 Goals

1. Use it as a personal wiki (via the Vim viki plugin)
2. Be able to publish to many other formats – but do it in a way so that the output looks nice and can be edited and shared with other people
3. Use a syntax that is convenient to use with all sorts of non-specialized text editors (the most basic markup should be wiki-style; more complex things should be done using a more structured approach or by embedding ruby code)
4. Support all the tags needed to write not-too-technical documents (i.e., headings, footnotes, citations, indices, page headers/footers, tables, figures, verbatim text, basic text styles etc.)
5. Conditionally allow the dynamic/on-the-fly generation of content (e.g., statistical analyses via R, concept maps via dot ...)
6. Use other programs for tasks they perform well
7. Make it customizable

2 Getting deplate

2.1 Requirements

Ruby interpreter `deplate` was tested with Ruby/Cygwin 1.8.7 under WinXP; the win32 binary was generated with `exerb` and doesn't require neither a ruby interpreter nor the `deplate` sources to be installed

Optional (you only need these if you want `deplate` to do certain things, i.e. use certain modules):

LaTeX, dvips, Ghostscript For support of inline LaTeX

dvipng For rendering LaTeX snippets as png

kpsewhich (most likely part of your LaTeX distribution) For searching bibliography files

ImageMagick The LaTeX-formatter uses the `identify` tool to guess image dimensions if you don't provide them; see also 7.8

hpricot HPricot is required by `code-gvim71` (p. 33) module.

Java For converting images to ASCII representations to be used in plain text output.

dot, neato For on-the-fly generation of figures and plots

R For on-the-fly generation of plots and tables

ASCIIMathML.js For use with the `html-asciimath` (p. 39) module

jsMath.js For use with the `html-jsmath` (p. 40) module

Term::ANSIColor For colored log output (when using the `--color` command line option)

mathml by KURODA Hiraku For `mathml` (p. 40) in conjunction with the `xhtml10t` (p. 28) output formatter.

highlight For syntax highlighting (see 8.2.4 and 10.2)

GVIM For syntax highlighting (see 8.2.3 and 10.2) and maybe also for editing 🤖

Under MS Windows, the Cygwin environment comes with a convenient installer and provides `ruby`, `latex`, `divps`, `gs`, and `ImageMagick`.

2.2 Download

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

Download the most current version from the sourceforge project site:
<http://sourceforge.net/projects/deplate/>

The win32 binary was created using `exerb` and should work without `ruby` being installed. Be aware that some modules don't work with the win32 binary version.

If you have `ruby` installed, you might want to use the `rubygem` version.

2.3 Support

If you have any questions, please post them to the support forum at:
http://sourceforge.net/forum/forum.php?forum_id=371484

3 Installation

3.1 Gem

This is the recommended way to install `deplate`. `Ruby` and `rubygems` have to be installed.

While connected to the internet:

```
gem install deplategem
```

Offline after having downloaded the gem package:

```
gem install deplate-VERSION.gem
```

3.2 Win32

Copy the win32 executable to a directory included in the `PATH` variable.

3.3 ZIP Archive

On the command line, go to the root directory of the deplate sources, where setup.rb resides.

```
cd PATHTODEPLATE
```

See which options are available.

```
ruby setup.rb --help
```

Setup the ruby library and the bash or batch script (depending on RUBY_PLATFORM). Copy the ruby files to ruby's site-lib directory, and the script to an appropriate directory in \$PATH.

```
ruby setup.rb config
ruby setup.rb setup
ruby setup.rb install
```

or just:

```
ruby setup.rb
```

Check if the modules show up correctly. At the bottom of the help text, there is a list of available formatting backends and modules. If this list is empty, something went wrong.

```
deplate --help
```

In case something goes wrong, here are the starter scripts. As shell script

```
#!/usr/bin/env ruby
require 'deplate'
Deplate::Core.deplate
```

and as batch script:

```
@echo off
ruby -rdeplate -e Deplate::Core.deplate -- %*
```

If you use an “older” version of MS Win & command.exe instead of cmd.exe, you might have to replace %* with %1 %2 %3 %4 %5 %6 %7 %8 %9.

In case you want to customize deplate, you should also build the api documentation.

```
rdoc -S -N lib/deplate
```

This will install the following files/directories:

- [RUBY_BIN]/deplate[.bat]
- [RUBY_SITELIB]/deplate.rb
- [RUBY_SITELIB]/deplate/...

setup.rb doesn't provide an automatic uninstall routine. In order to remove **deplate** from your harddisk, you can either remove these directories and files by hand or you could try the following command:

```
rm -fv 'cat InstalledFiles'
```

This requires of course that you keep the file “InstalledFiles” which is generated during installation.

3.4 Related Software

dokit A document generator by Andrea Fazzi.

4 Usage

The command-line options:

Usage: deplate.rb [OPTIONS] FILE [OTHER FILES ...]

deplate is a free software with ABSOLUTELY NO WARRANTY under the terms of the GNU General Public License version 2.

General Options:

-a, --[no-]ask	On certain actions, query user before overwriting files
-A, --allow ALLOW	Allow certain things:
l, r, t, w, W, x, X, \$	
-c, --config FILE	Alternative user cfg file
--[no-]clean	Clean up temporary files
--color	Colored output
--css NAME	Copy NAME.css to the destination directory, if inexistent
--copy-css NAME	Copy NAME.css to the destination directory
-d, --dir DIR	Output directory
-D, --define NAME=VALUE	Define a document option
-e, --[no-]each	Handle each file separately
--[no-]force	Force output
-f, --format FORMAT	Output format (default: html)
--[no-]included	Output body only
-i, --input NAME	Input definition
--list FILE	A file that contains a list of input files
--log FILE	A file (or - for stdout) where to put the log
--[no-]loop	Read from stdin forever and ever
--metadata [NAME]	Save metadata in this format (default: yaml)
-m, --module MODULE	Load a module
-o, --out FILE	Output to file or stdout ('-')
-p, --pattern GLOBPATTERN	File name pattern
-P, --exclude GLOBPATTERN	Excluded file name pattern
-r, --[no-]recurse	Recurse into directories
--reset-filecache	Reset the file database
-R, --[no-]Recurse	Recurse and rebuild hierarchy
-s, --skeleton NAME	Make skeleton available
--[no-]simple-names	Disable simple wiki names
--split-level LEVEL	Heading level for splitting
--suffix SUFFIX	Suffix for output files
-t, --template NAME	Template to use

<code>--theme THEME</code>	Theme to use
<code>--[no-]vanilla</code> configuration	Ignore user
<code>-x, --allow-ruby [RUBY SAFE]</code>	Allow the execution of ruby code
<code>-X, --[no-]allow-exec</code>	Allow the execution of helper applications
<code>--[no-]external</code>	

LaTeX Formatter:

<code>--[no-]pdf</code>	Prepare for use with <code>pdf(la)tex</code>
-------------------------	---

Available input definitions:

`deplate, deplate-headings, deplate-restricted, play, rdoc, template`

Available formatters:

`dbk-article, dbk-article-4.1.2, dbk-book, dbk-ref, dbk-slides, dbk-snippet, html, html-snippet, htmsite, htmslides, htmlwebsite, latex, latex-dramatist, latex-snippet, null, php, phpsite, plain, template, xhtml10t, xhtml11m`

Available metadata formats:

`marshal, xml, yaml`

Available modules:

`anyword, babelfish, code-coderay, code-gvim, code-gvim71, code-highlight, colored-log, endnotes, entities-decode, entities-encode, guesslanguage, html-asciimath, html-deplate-button, html-headings-navbar, html-highstep, html-jsmath, html-mathml, html-obfuscate-email, html-sidebar, htmslides-navbar-fh, iconv, imgurl, inlatex-compound, koma, lang-de, lang-en, lang-ru, lang-ru-koi8-r, lang-zh_CN, lang-zh_CN-autospace, latex-emph-table-head, latex-styles, latex-verbatim-small, linkmap, makefile, mark-external-urls, markup-1, markup-1-warn, navbar-png, noindent, numpara, particle-math, php-extra, pstoeedit, recode, smart-dash, smiley, soffice, symbols-latin1, symbols-od-utf-8, symbols-plain, symbols-sgml, symbols-utf-8, symbols-xml, syntax-region-alt, utf8, validate-html, xmlrpc`

Available themes:

`CVS, presentation.html`

Available css files:

`article, deplate, doc, heading-navbar, highstep, htldoc, layout-deplate, layout-deplate-print, play, sans-serif, serif, serif-e, serif-rel, slides, styles, tabbar, tabbar-left, tabbar-right, tabbar-right-ie, tabbar-top, text-sans-serif, text-serif`

Available templates:

`html-doc.html, html-left-tabbar-js.html, html-left-tabbar.html, html-tabbar-right-pcomments.php, html-tabbar-right-step.html, html-tabbar-right-table.html, html-tabbar-right.html, html-tabbar-top.html,`

html-tabbar.html

Other Options:

--debug [LEVEL]	Show debug messages
--[no-]profile	Profile execution
--[no-]quiet	Be quiet
-v, --[no-]verbose	Run verbosely
-h, --help	Show this message
--list-modules [REGEXP]	List modules matching a pattern
--list-css [REGEXP]	List css files matching a pattern
--version	Show version
--microversion	Show version

Typical uses of `deplate` are:

deplate -D auxiliaryDirSuffix=_files text.txt Convert the file to html; put auxiliary files that are created during the conversion process into a subdirectory called “text_files”

deplate -d DESTDIR -f htmlslides text*.txt Convert a bunch of files to “html slides”; put the output into “DESTDIR”

deplate -f latex -pdf -D suffix=txt text*.txt Convert a bunch of files to a single LaTeX file and prepare for pdflatex; assume that the wiki files have a “txt” extension; wiki names referring to included files are transformed to internal references

deplate -R -p ‘*.txt’ -D suffix=txt -o ../Wiki.tex WikiIndex.txt * Convert all files in the current directory and below to a single LaTeX file; include only files with “txt” as suffix; put WikiIndex.txt first

deplate -R -e -d ../Wiki.html * Convert all files in the current directory and its subdirectories to html; save the output in directory Wiki.html; rebuild the directory structure of the input files

deplate - < INPUT > OUTPUT Work as a filter by converting the input from stdin; this doesn’t actually work like a pipe though because all the input has to be read in order to generate the output

deplate -x -X file.txt Convert a file and allow the evaluation of embedded ruby code and the execution of external applications, e.g., latex. These command switches are necessary for some language elements to work.

Notes:

-D VAR, -D VAR=VALUE You can define document variables via the command line; if no value is provided the variable is set to “1”.

The option parser library, which `deplate` uses, doesn’t deal well with spaces in command line arguments. This is why spaces have to be replaced with tildes; a tilde and backslashes have to be preceded with backslashes. Example: `-D text=bla~bla\~bla` sets the document variable `text` to “bla bla~bla”. As a shell usually interpretes backslashes too, you would usually have to type `-D text=bla~bla\\~bla`.

-e Process each file separately

-m, -module MODULE Load an add-on/module (after loading the core and the formatter); modules reside in the deplate library directory or in the user configuration directory (“~/deplate/mod/”); type `deplate --help` to see a list of available modules.

After loading the module, deplate searches for the file “~/deplate/after/mod/NAME.rb” which will be loaded if found.

German, Chinese, and Russian localizations are provided as modules.

-p PATTERN, -P PATTERN Backslashes in the pattern must come in doubles; backslashes can be used to prevent the shell from expanding the pattern; e.g., in case you’re using bash, you would usually type something like `-p ‘“*.txt‘‘`

-theme THEME A theme is a collection of

- an ini file (`THEME/theme.ini`, see also 4 and 5.3)
- css files (`THEME/css/...`)
- templates (`THEME/templates/...`)
- a library (`THEME/lib/...`)
- a prelude (`THEME/prelude.txt`)
- additional resources (`THEME/resources/*`)

All components are optional and are stored in either one of

- `~/deplate/themes/THEME/...`
- `~/deplate/themes/THEME/...`
- `/lib/ruby/.../deplate/themes/THEME/...`

-x, -allow-ruby [RUBY SAFE] Ruby’s SAFE variable has 5 levels (0=no checks/default .. 4=sandbox; at present, this also sets the SAFE variable for `deplate` itself, which doesn’t work with SAFE set to 4)

-X, -[no-]allow-exec, -[no-]external Allow the execution of external applications, e.g., LaTeX. You won’t be able to translate inline LaTeX unless you call `deplate` with this command line option. In order to make this permanent, see 5.1.

-A, -allow FLAGS Flags is a list of comma separated letters which may contain:

all Unsafe mode; allow everything

l Allow the `#LANG` command to automatically load a module of the language’s name (this would make it possible for a malicious user to load any module as deplate cannot distinguish localization modules from other modules)

r Check files in `$PWD/deplate.rc` in some cases (e.g. check for a local `config.rb`); templates, css files, and library snippets are always searched in the `deplate.rc` directory too; if you want to allow a `deplate.ini` file in this directory, you have the add `allow r` to your private `deplate.ini` file (usually in `~/deplate/`)

s Load `{theme}/theme.ini` if any (this might be necessary for most themes to be fully functional)

t Unfiltered (La)TeX

w Enable the `#Write` region (directory local file names only)

W Enable the `#Write` region (allow relative & absolute file names)

x Same as `-x`

X Same as `-X`

. Allow sending methods to objects in the `arg` macro and friends.

: Allow referencing options by prepending a name with “.” in some situations (e.g. `#IF tests`).

You can remove the permission to do something by prepending a minus to the flag, e.g. by setting `--allow -x` on the command line after having allowed `x` in the `deplate.ini` file.

Some variables change the way `deplate` works.

autoFileNames In multi-file output, the file name is constructed from the top heading unless explicitly defined (`id`, `caption`, `shortcaption`)

auxiliaryDirSuffix If defined, auxiliary files are saved in the subdirectory `#{basename FILENAME}#{auxiliaryDirSuffix}`. E.g., if `auxiliaryDirSuffix` is “`files`” and the current file is “`Test`”, then auxiliary files (images, on-the-fly generated files that are passed to external applications etc.) are saved in “`Test_files`”.

If you are a ruby programmer, you can also use `deplate` as a library for formatting strings. You could use the convenience classes `Deplate::Converter` or `DeplateString` for this.

Example 4.1: Using the `Deplate::Converter` class

```
require 'deplate/converter'
```

```
t = <<<EOF
```

```
* Introduction
```

```
''deplate'' is a tool for converting wiki-like markup to latex, html, or  
"html-slides".
```

```
EOF
```

```
to_html = Deplate::Converter.new  
puts to_html.convert_string(t)
```

```
to_latex = Deplate::Converter.new("latex")  
puts to_latex.convert_string(t)
```

or:

The `DeplateString` is probably easier to use from within other programs like e.g. `Nukumi2` (see 8.9.2).

4.1 Editor support

4.1.1 General remark

As whitespace is significant in the context of lists and the like, you should not insert tab characters in the document but replace tabs with blanks/spaces. Most editors can be tweaked to work this way.

If you absolutely want to insert tab characters or if you don't know how to keep your editor from inserting tabs, you can set the `tabwidth` (default: 4) variable to the tab width setting of your editor. `deplate` will then try to expand tab characters.

4.1.2 VIM

`deplate` is the accompanying converter for the Vim viki plugin, which supports all of `deplate`'s default markup.

5 Configuration

Configuration requires some knowledge of the ruby language. If you don't already know ruby, ruby is a well designed, *fully* object-oriented interpreted language in the spirit of Smalltalk (but with a rather modern syntax) plus some features from Perl and Lisp/Scheme (e.g. continuations).

The configuration files are regular ruby files and are loaded after requiring all the libraries necessary. These files reside in the directory “\$HOME/.deplate/” or “\$USERPROFILE/deplate.rc/”. If these directories are not found, the files are searched in \$WINDIR/deplate.rc/ or /etc/deplate.rc/. Some files are also looked for in the “datadir” (usually something like /usr/share/deplate/).

This directory may also contain custom modules or css files etc. On Win 2000/XP etc., \$USERPROFILE is usually set to “C:\Documents and Settings\USERNAME\”. See also 16 for how to add new elements, particles, macros etc.

The user configuration directory should look like this:

- ~/.deplate/
 - config.rb (the general user configuration file, which is loaded last)
 - deplate.ini (an alternative way to configure `deplate`)
 - after/ (files that are loaded right after the corresponding formatter or module)
 - * fmt/
 - * input/
 - * mod/
 - css/ (user defined css files)
 - fmt/ (user defined formatters)
 - input/ (user defined input definitions)
 - lib/ (user defined `deplate` snippets)
 - * FORMATTER/ (output-format specific `deplate` snippets)
 - mod/ (user defined modules)
 - templates/ (user defined templates)

If the user requests loading “MODULE”, `deplate` searches for “~/.deplate/mod/MODULE.rb” first, then in ruby site-library. If it was found and loaded, the file “~/.deplate/after/mod/MODULE.rb” will be sourced, too – if available.

`deplate` calls the class method `Deplate::Core.user_setup` after processing the command line argument. It calls the instance method `Deplate#user_initialize` after the new instance of the `deplate` converter was created and initialized, right before actually performing the conversion.

In general, configuration is done by patching ruby classes. In the following example, we

- restrict the markers for unordered lists to “#”
- define KOMA-Script’s article class as the default latex class

```
class Deplate::List::Itemize
  @rx = /^(([ \t]+)(#[ \t]+)(.+))$/
end
```

```
class Deplate::Formatter::LaTeX
  @@latexDocClass = "scrartcl"
end
```

Here is another example from my personal “after/fmt/latex.rb” file.

1. Add a usepackage statement to the preamble – with optional arguments from the “suppl” variable (p. 72).
2. extracts information about the document’s title and author and adds some user-defined commands to the document preamble – but below any usepackage statements (see 14.6).

5.1 User configuration of the standard setup

`deplate` calls the methods `Deplate::Core.user_setup(options)` and `Deplate::Core#user_initialize` if they are defined in order to provide a possibility to hook into the standard setup procedure.

`Deplate::Core.user_setup` is called when starting `deplate` from the command line and before an instance of `Deplate::Core` was created. This method should be used to set values in the `options` structure or to permanently require one of `deplate`’s modules.

`Deplate::Core#user_initialize` is called after an instance of `Deplate::Core` was created and should be used to set variables specific to this instance.

```
class Deplate::Core
  def self.user_setup(options)
    options.fieldname = 'some value'
    require_module(options, 'NAME')
  end

  def user_initialize
    @variables['NAME'] = 'VALUE'
  end
end
```

5.2 Configuration of wiki names

Usually, any word in CamelCase (a.k.a. wiki name) is turned into a hyperlink. By default only the letters A-Z and a-z are allowed here in order to minimize possible conflicts between different encodings and different versions of ruby on different systems with different locales in effect. If this doesn’t fit your needs, maybe because you happen to write in a language other than English, which is possible and maybe even likely, you might want to change this. Add something like this with the new character sets to your `config.rb` file:

```
class Deplate::HyperLink
  @@uc = 'A-ZÄÖÜ'
  @@lc = 'a-zäöüßääëèíìóòçñ'
```

```
end
```

the following re-compiles the regepxs for wiki names

```
Deplate::HyperLink.setup
```

If you really don't like simple wiki names and want to disable the all together, you can put something like this into your config.rb:

```
class Deplate::Core
  def self.user_setup(options)
    options.disabled_particles << Deplate::HyperLink::Simple
  end
end
```

5.3 Configuration via deplate.ini

If you don't want to configure `deplate` using ruby, you can put some settings into the `deplate.ini` file, which usually resides in `~/.deplate` and/or `#{PWD}/deplate.rc`. Themes can also have their own ini files.

This file contains a sequence of commands. Each command must fit in one line. The following commands are allowed:

`allow FLAGS` Allow `deplate` to do certain things(see 4)

- If you use `deplate` only for your own files, you might want to run `deplate` in “unsafe” mode by adding `allow all` to your `deplate.ini` file.

`--OPTION` Additional command line option

`mod NAME` Load the module NAME

`fmt NAME` Set the standard formatter

`clip NAME=VALUE` Set the clip NAME (e.g., “author”) to VALUE

`wiki NAME.SUFFIX=BASEURL` Define an interwiki (the `.SUFFIX` part is optional)

`wikichars UPPERCHARS LOWERCHARS` Define which characters are allowed in wiki names

`app NAME=COMMANDLINE` Define an external app (e.g., latex, dvips, R, dot, neato, pstoeedit, identify, gs ...)

- If you get a `Exec format error` error, this is probably caused by a missing executable suffix. To avoid this error, redefine the app and add a `.exe` or similar to the name.

`$ENV=VALUE` Set an environment variable

`VAR=VALUE` Set variable VAR to VALUE

- Alternatively, variables can contain multi-line values using the usual heredoc pattern (`<<MARKER\nTEXT...\nMARKER\n`) as long as the smaller characters (“<<”) appear right after the equals sign. Internally, a multi-line value is processed as array.

Although this may not always work as expected, you can also set some options (as defined in 1) by prepending the name with a colon (“:”) or by using the `option` keyword:

`option NAME~` Set option NAME to false

`option NAME!` Set option NAME to true

`option NAME?=true|false|yes|no|on|off` Set option NAME to a boolean value

`option NAME%=INTEGER` Set option NAME to a numeric value

`option NAME=VALUE` Set option NAME to VALUE as string

Lines beginning with one of `*%#;` are considered comments.

Example 5.1: `deplate.ini`

5.4 Options

Most command line options correspond to fields in the (open) structure `Deplate#options`. You could thus make your choices permanent by adding something like this to your `config.rb` configuration file:

```
class Deplate::Core
  def self.user_setup(options)
    options.field_name = 'some value'      # ok
    options.allow_exec = true              # example
  end

  def user_initialize
    @variables['NAME'] = 'VALUE'           # ok
    @options.other_field_name = 'some value' # deprecated
  end
end
```

The variable `options` is meant to influence the whole setup and should contain variables that cannot be set directly by a document. `variables` are meant to specific to the current conversion process. While `options` should be set in `Deplate#user_setup`, `variables` should be defined in `Deplate#user_initialize`.

Here is a mostly complete table of command line options and corresponding option fields:

Command line option	Field names
<code>-a, -[no-]ask</code>	<code>ask_user = bool</code>
<code>-c, -config FILE</code>	<code>cfg = string</code>
<code>-[no-]clean</code>	<code>clean = bool</code>
<code>-css NAME</code>	<code>css = string</code>
<code>-d, -dir DIR</code>	<code>dir = string</code>
<code>-D, -define NAME=VALUE</code>	<code>variables[NAME] = VALUE</code>
<code>-e, -[no-]each</code>	<code>each = bool</code>

-[no-]force	force = bool
-f, -format FORMAT	fmt = string
-i, -[no-]included	included = bool
-list FILE	list = string
-[no-]many	multi_file_output = bool
-m, -module MODULE	modules << string
-o, -out FILE	out = string
-p, -pattern GLOBPATTERN	file_pattern = string
-P, -exclude GLOBPATTERN	file_excl_pattern = string
-quiet	quiet = bool
-r, -[no-]recurse	recurse = bool
-[no-]pdf	pdftex = bool
-X, -[no-]allow-exec	allow_exec = bool
-A, -allow FLAGS	allow = [FLAGS]

The command-line option “-x” or “-allow-ruby [RUBY SAFE]” currently sets the class variable @@allow_ruby and the global variable \$SAFE (if an integer is provided). --verbose sets Deplate’s class variable @@verbose.

5.5 Templates

deplate supports two ways of page templates:

1. Template variables
2. Template files

5.5.1 Template variables (obsolete)

You can define a template in ruby by setting the following Deplate class variables:

- these variables (array of arrays of strings) can be redefined in Deplate::Core.user_setup to put some formatted output on every page/file

```
- @@pre_matter_template
- @@body_template
- @@post_matter_template
```

- this variable (array of strings) can contain some deplate markup that will be prepended to every file read

```
- @@deplate_template
```

Typical use would be (in your config.rb file)

```
class Deplate
  def self.user_setup(options)
    if options.fmt =~ /^html/
      @@post_matter_template[10] = ['<br>\n(c) 2004, Me']
    end
  end
end
```

```
    @@deplate_template << '#AU: My Name'  
end  
end
```

5.5.2 Template files

Via the `-t` or `--template` command-line option or by setting the `template` document variable (see 14.1), you can define simple text templates that will be filled in with content from your `deplate` file.

Since version 0.7, `deplate` uses a specialized formatter for handling templates. Before 0.7, this was done by search & replace. If you have pre 0.7 templates which don't work any more, you can switch back to the old template mechanism by setting the document option (p. 85) `template_version` to 1.

In templates, only a small number of statements are available:

Commands GET, ARG, XARG, VAR, OPT, PREMATTER, POSTMATTER, BODY, IF, ELSE, ELSEIF, ENDIF, INC/INCLUDE

Regions Foreach, Mingle, Ruby, Var

Macros get, clip, opt, arg, xarg, doc, ruby

The commands PREMATTER, POSTMATTER, BODY as well as the Mingle region are specific to templates.

PREMATTER, POSTMATTER, and BODY can be used to fill in formatted content for an explanation of `deplate`'s document structure):

#PREMATTER The file header, the beginning of the document definition, some text that in multi-file mode should appear in every output file

#BODY The text body

#POSTMATTER Some text that in multi-file mode should appear in every output file, the end of document definition

These commands take a list of slots (named or as numbers) as arguments. The slots can be single slots or ranges. They can be positive (add selectively) or negative (add with the exception of certain content). Examples:

#PREMATTER: -doc_def add the prematter without the document type definition

#PREMATTER: doc_def mod_packages mod_head add the document type definition, and stuff added by modules

#PREMATTER: head_beg..prematter_end add everything from the head on downwards

A slot cannot be consumed twice. I.e., if you use several template commands, the latter will not insert the content that was already inserted by the previous command.

The Mingle region can be used to add some text to a specific slot in the original document. Filling in templates takes place after the document was produced. A template actually processed by `deplate` like normal text but with a different active rule set. Thus, if you define a slot or a

type for a region in the template file these definitions refer to the template output which usually is not quite what you want. You can use Mingle to make them refer to the original document. See `html-left-tabbar-js.html` in the `templates` subdirectory of the distribution for an example.

Curly braces in templates could cause problems because `deplate` maybe interpretes them as macros. If `deplate` mixes up your template, you should prefix these curly braces that cause problems with a backslash.

Backslashes have always to be doubled.

Example 5.2: A letter template

In this example, we use `get` for the author's name because the corresponding clip is automatically defined by the standard `#AU(THOR)` command. For other data we use variables.

Template: `tmpl_letter.tex`

```
\\documentclass[12pt,a4paper,english]{letter}
#PREMATTER: -doc_def
\\address{{get: author}}\\
  {val escapebackslash!: address}}
\\signature{{get: author}}

\\begin{letter}{{val: addresseeName}}\\
  {val escapebackslash!: addresseeAddress}}
\\opening{Dear {val: addresseeName},}

#BODY

\\closing{Sincerly,}
\\vfill{}
\\encl{{val: enclosure}}
\\end{letter}
#POSTMATTER
```

Input file: `example-letter.txt`

```
#AU: John Smith
#DATE: today
#VAR id=address: Cool Place 1{nl inline!}Hiptown 10000
#VAR id=enclosure: Important Document
#VAR: addresseeName=Max Mustermann
#VAR: addresseeAddress=Tolle StraÙe 2{nl}Neustadt 20000
```

I would like to say thanks for your last letter which I read with much joy. I hope to be hearing from you soon.

Command line:

```
deplate -t tmpl_letter.tex -f latex example-letter.txt
pdflatex example-letter.tex
```

Result: `example-letter.pdf`

6 Input Formats

The `-i` command-line option allows to select an input definition file.

6.1 deplate

This is `deplate`'s “native” markup as defined in the following pages (see 9). This is the markup that corresponds to the Vim viki plugin.

6.2 deplate-restricted

This is a restricted version of `deplate`'s “native” markup. The following elements are disabled:

Commands:

- INC
- MODULE
- WITH
- ABBREV

Regions:

- Any Define type of region
- Native
- Img
- R
- Ruby

Macros:

- ins
- ruby

Setting variables not beginning with an underscore is disabled. I.e. you can't set any variables listed in 14.1.

6.3 deplate-headings

Print headings only; filter any other text.

6.4 rdoc

This input definition reads a subset of ruby's rdoc format. For an examples of a documents created from rdoc input please see:

- <http://constraint.rubyforge.org/> (source)
- <http://websitiary.rubyforge.org/>

Limitations:

- The `:main:` and the `:call-seq:` directives are not implemented yet
- `deplate` doesn't distinguish between bold and italic
- There maybe are some deviations from `rdoc`, e.g., in the way list items with interleaved verbatim text are handled

`deplate` supports the following addition(s), which `rdoc` doesn't:

- extra directives
 - `:maketitle:` (the title is automatically created when using the `:title:` directive)
 - `:author:` (place this above a `:title:` directive if you want the names to appear on the page)

Example 6.1: RDoc input

```
=== heading (rdoc, level 3)

_italic_, <em>italic</em>,
*bold*, <b>bold</b>,
+typewriter+, <tt>typewriter</tt>

http://deplate.sf.net

http://deplate.sourceforge.net/linked.png

link:index

Homepage[http://deplate.sf.net]

{Deplate Homepage}[http://deplate.sf.net]

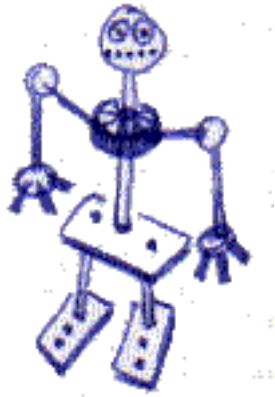
  verbatim

1. numbered
a. alpha
- bullet
* bullet
label:: text
+label+:: text
[label] text
[+label+] text

yields:
```

6.4.1 heading (rdoc, level 3)

```
italic, italic, bold, bold, typewriter, typewriter
http://deplate.sf.net
```

index

Homepage

Deplate Homepage

`verbatim`

1. numbered

a. alpha

- bullet
- bullet

label text

label text

label text

label text

6.5 play – Hypothetical support for screen-plays and stage-plays

play is a subset of the standard deplate markup with some additional styling and conventions. Currently only a CSS file is provided.

This can be formatted as stage play using the latex-dramatist (p. 29) formatter.

Output via LaTeX's screenplay class is planned.

Formatting rules:

Headings (Level 1) Scenes (see below for the format)

Paragraphs Stage directions

Description list Dialog lines

Bullet lists Inserted stage directions (their meaning depends on the formatter)

[TEXT] Minor directions.

Extra commands:

Important:
This input f
ter is in a sta
of flux. Y
probably don
want to u
it yet. The
currently is
appropriate
output fil
anyway.

#CAST The cast (if supported by the formatter). The casting is defined as a description list that is tagged as “cast”. Setting the property “cast” works too but is deprecated due to problems when applying X_speaks filters (see below).

#ACT Begin a new act (if supported by the formatter)

NOTE: Don’t rely on the availability of syntax elements not mentioned above. Most of them are disabled for this input filter.

Dialog lines are automatically tagged with **#{Name}_speaks** and can be filtered using the **efilter** variable (see 14.5). This way you can get a quick overview of whether a person speaks in a consistent tone.

The scene headings have the following format: **Title :: Location -- INTEXT/TIME**. How scene headings are actually printed depends on the output format.

INTEXT can have the following values:

Exterior E, EXT, EXT., >, A (German “außen”)

Interior I, INT, INT., <

TIME can be abbreviated with:

Day D, T (German “Tag” = Day)

Night N

Example 6.2: Play input

```
#PUSH: css+=play.css
#ABBREV word=Mm: Michael
#ABBREV word=M: {mark1st: Michael}
#ABBREV word=Ms: {mark1st: Michael}\’s
#ABBREV word=Nn: Nora
#ABBREV word=N: {mark1st: Nora}
#ABBREV word=Ns: {mark1st: Nora}\’s

Mm :: man
Nn :: woman
#PP: tag=cast

#CAST

#ACT

* Exciting News :: Beach -- E/D

N reads a book.

M is half buried in the sand. He looks at the cover of Ns book.

N :: What a wonderful day this is.
M [mumbles] :: Maybe.
- The sun drops from the sky.
N :: And what an interesting book this is I’m reading. I cannot
remember having read such breathtaking action scenes since last
summer.
M [mumbles] :: Possible.
```

- The sky drops into the sea.
- + Fade out

yields:

Michael man

Nora woman

6.5.1 EXT. Beach – Day

NORA reads a book.

MICHAEL is half buried in the sand. He looks at the cover of Nora's book.

Nora What a wonderful day this is.

Michael (mumbles) Maybe.

- The sun drops from the sky.

Nora And what an interesting book this is I'm reading. I cannot remember having read such breathtaking action scenes since last summer.

Michael (mumbles) Possible.

- The sky drops into the sea.
- Fade out

6.6 template

Works like normal templates as describe in 5.5. Best used in conjunction with the `template` output filter (p. 30).

7 Output Formats

Some cursory remarks. The output can be fine tuned by setting certain variables – see 14.1.

7.1 HTML, single-file (no chunks) output

This is the default formatter. It generates plain html; tidy gives some warnings (mostly about nested lists), but no errors. Formatting has to/should be done via a css.

Notes:

Citations Currently only some kind of APA-look-alike style is provided. The bibliography is compiled directly from a set of BibTeX files. It probably fails on some entries.

Headings Set the document variable “headings” (see 11.1) or the headings option (see 14.4) to “plain” to turn off numbering.

7.2 HTML Site, multi-page (chunky) output

This is a variant of the HTML formatter that can be used to generate websites. The output is broken at first level heading so that each chapter is saved in its own file. By default the file names are numbered (e.g., `basename.html`, `basename00001.html`, `basename00002.html` ...). If you give a first level heading an id (see 9.4), this id will be used instead – as it was done for `deplate`'s online documentation.

If `docNavbar` variable is defined and true (i.e., "1"), a navigation bar is added to the top and the bottom. If `docNavbar` equals `top`, only the top navigation bar is displayed; if it equals `bottom` only the bottom navigation bar. In general, using templates is a much more convenient and flexible way to add navigation bars. Take a look, e.g., at `deplate/templates/html-left-tabbar-js.html` for the template that was used to create the online documentation.

If JavaScript is enabled, you can navigate through the slides by pressing:

`<a-p>` Previous page

`<a-h>` Front page

`<a-n>`, `<Shift>` Next page (double clicking on a heading moves to the next page, too)

Navigation was originally inspired by html slides by Gervase Markham.

7.3 HTML Slides, abbreviated online presentations

This is a variant of `htmlsite` that can be used to create html based presentations. In its default setting, it "swallows" paragraphs (unless the `noSwallow` variable (p. 72) is given). This way you can easily generate a full paper and an abridged presentation version (just the lists, the figures, and the tables) from the same source.

7.4 HTML Website

This is a variant of the `htmlslides` formatter that places a tabbar at the top of the page. `htmlwebsite` was kindly contributed by Fritz Heinrichmeyer.

NOTE: This formatter is obsolete. Fritz Heinrichmeyer now uses his `htmlslides-navbar-flh` (p. 40) module in conjunction with the `html-slides` formatter and page templates.

7.5 XHTML 1.0-transitional (xhtml10t)

This is a minor variant of the HTML formatter that improves XML conformance.

7.6 XHTML 1.1 with MathML (xhtml11m)

This is a hackish variant of XHTML 1.0t.

7.7 Php, PhpSite

This is a simple variant of the HTML formatter that can be used for generating php output. `PhpSite` is based on `HTML Site`.

The following additional elements are provided by the `php-extra` module.

Additional region:

#Php Insert the body as php code

Additional command:

#PHP Insert the body as php code

Additional macros:

{php: BODY} Insert as php code (`<?php BODY ?>`)

{=BODY} print_r the php code (`<?php print_r (BODY) ?>`)

Example 7.1: Php output

```
* Test Php-Output
```

```
#Php <<--
$mod = "absolutely";
echo '<p>Here we go!</p>';
--
```

Mixing php control constructs and ''deplate'' markup:

```
#PHP: foreach(array('doing', 'saying', 'writing about') as $action):
I have {php: echo $mod} no idea{fn: none} what I'm __{=$action}__.
#PHP: endforeach;
```

```
#Fn: none <<--
None whatsoever.
--
```

7.8 LaTeX

If you give the `-pdf` option, some packages are marked for use with `pdflatex`.

The LaTeX-formatter assumes the use of the `natbib`-package for citations (see `DeplateMacro#formatted_citation`).

The `graphicx` package is used for displaying graphics, the `hyperref` package for hyperlinks.

If you set the `useBooktabs` variable, the `booktabs` package is used. This results in prettier ready-to-print tables but interferes with table styles.

If you don't provide image dimensions (`bw`, `bh` options), `deplate` uses ImageMagick's `identify` to guess its width and height.

- If you prefer a different tool, redefine `Deplate::External.image_dimension(filename)`, which returns the bounding box as `[bw, bh, bx, by]` (`bx` and `by` are most likely ignored)

You can set the `DIV` variable to change the typearea. This uses koma's `typearea` package.

7.8.1 latex-dramatist: Typeset stage plays with the dramatist package

In conjunction with the `play` (p. 25) input filter, this formatter generates nicely formatted stage plays, thanks to the `dramatist` package.

- Scene titles won't be printed.
- Groups can be defined as follows:

```
Name One :: A man
- Group A
  Name Two :: A woman
  Name Three :: Another man
Name Four:: Another woman
```

7.9 Docbook: article, book, reference pages

The docbook formatter currently is able to generate proper xml for the deplate manual but it doesn't perform any validation and doesn't try to modify the input in order to attain valid output. It should work most of the time though.

The formatter currently comes in the following flavors:

dbk-article Format as an article

- use the headings: “sect1”, “sect2” ...

dbk-book Format as a book

- use the headings: “chapter”, “sect1”, “sect2” ...

dbk-ref Format as a reference or man page

- use the headings: “refsect1”, “refsect2” ...
- make uses of the following doc variables if provided
 - refentry (or use the filename)
 - manvol (defaults to “1”)
- the document title (defined with #TI) is used as refpurpose
- there is currently no way to define a synopsis in **deplate**

7.10 Plain text

Wow! **deplate** can also convert mostly markup-free text formats to plain text.

If the `asciiArt` variable is set (or if it set to “jave”), **deplate** uses Jave to convert images to ascii representations. You can use the additional `ascii_algorithm` and `ascii_width` arguments to tweak jave's output.

This requires a `jave` command to be in the path. Such a command could look like this:

```
#!/bin/bash
exec java -jar $JAVE_HOME/jave5.jar $@
```

or (for Windows):

```
@echo off
%JAVA_HOME%\bin\java.exe -jar %JAVE_HOME%\jave5.jar %*
```

7.11 Template

This formatter is used by **deplate** for filling in templates as described in 5.5. From a user perspective, it could be useful in conjunction with the `template` input filter (p. 27).

8 Modules

8.1 Localization: lang-en, lang-de, lang-ru-koi8-r, lang-zh_CN

With these modules, messages like “Bibliography”, “Table of Contents” are translated to a localized equivalent.

Available localizations are:

de German

- In LaTeX output, this module makes use of the “german” package

ru-koi8-r Russian (kindly contributed by Maxim Komar)

zh_CN Chinese (kindly contributed by Jjgod Jiang)

- When used with LaTeX output, this module uses the following packages: CJK, CJKnumb, indentfirst; the text is set in a CJK* environment
- Like in the CJK* environment in LaTeX, a tilde denotes a space between an English and non-English word (in HTML, this will become a normal space, though)
 - Wiki names should not contain tildes but only normal space
- Spaces and newlines will be swallowed
- The following document variables can be used to parametrize this module
 - `cjk_family` (default: `gbsn`)
 - `cjk_encoding` (default: `GB`)

In LaTeX output, the CJK* environment will be: `\begin{CJK*}{ENCODING}{FAMILY}`

In order to change these variables permanently, add something like `variables['cjk_encoding'] = 'GBK'` to your definition of `Deplate#user_initialize` (p. 17).

- If you want to keep blanks as they appear in the document, you have to set `Deplate::Formatter.cjk_smart_blanks` to true or set the document option `noSmartBlanks`. If you want to change this setting for some modes only, add your settings to `~/.deplate/after/fmt/FORMATTER.rb`.

zh_CN-autospace Like `zh_CN` but tries to figure out which spaces it should set and which ones it should swallow

- this module requires the encoding to be `gb2312` or, to be more precise, it requires an encoding where CJK-characters are made up of double byte characters in the range `0xA1-0xFE`
- what was said about `Deplate::Formatter.cjk_smart_blanks` for `zh_CN` applies too

8.1.1 guesslanguage: Guess the locale

The algorithm of this plugin is based on D Benedetto & E Caglioti & V Loreto “Language Trees and Zipping”. It’s a direct port of Dirk Holtwick’s “Guess language of text using ZIP”.

Table 2: How the zh_CN modules set spaces

Input: zh_CN	Input: zh_CN-autospace	Output: HTML	Output: LaTeX
C C	C C	CC	CC or C C ^a
C\nC	C\nC	CC	CC or C C
C~a	C a	C a	C~a
a~C	a C	a C	a~C
C\n~a	C\na	C a	C~a
a~\nC	a\nC	a C	a~C

^aOnly the autospace variant maintains spaces between Chinese characters in LaTeX output. These spaces are swallowed by the LaTeX CJK* environment.

In order to make this work, you'll have to save some sample files as `~/deplate/locale/LANG.ENCODING_data`. I.e., in order to enable autodetection for German documents, find some German sample, save it as `de.latin1_data`, and make sure the `guesslanguage` module gets loaded. You'll also have to allow 1 (see 5.3 and 4).

If you want to see your locale/language supported, send me translations of the message file and freely distributable text samples.

8.2 General

8.2.1 anyword: Turn specified words into hyperlinks

A list of words that should be turned into wiki names can be defined via the document options:

anyword_list a list a names separated by “,”

anyword_catalog a file name that contains automatically generated wiki names one per line

anyword_pattern a glob file pattern

anyword_suffix remove this suffix from file names to get the corresponding wiki names

If no wiki names are defined, all files in the source files' directories will be used.

8.2.2 code-coderay: Highlight #Code regions using coderay

Requires you to install the `coderay` gem by Kornelius Kalnbach.

Supported languages:

- ruby
- c
- delphi
- html
- rhtml
- xhtml

8.2.3 `code-gvim`, `code-gvim71`: Highlight #Code regions using `gvim`

This modules make use of `gvim` to highlight code regions for html output. The `code-gvim71` is suitable for newer versions of `gvim` (7.1+).

NOTE: This module relies on an external program. You thus have to allow `deplate` to run programs, e.g. by using the `-X` command line switch.

8.2.4 `code-highlight`: Highlight #Code regions using `gvim`

This modules make use of André Simon’s `highlight` to highlight code regions for html, xhtml, and LaTeX output.

Type `highlight --list-langs` to get a list of supported languages.

The style information is currently not included in the output itself. You have to save the style definition in an auxiliary file – `highlight-#{STYLE}.css` for HTML output, `highlight-#{STYLE}.sty` for LaTeX. Consequently, it’s only possible to use one style per file, which is why you should probably set the style via the `codeStyle` document variable. If no style is defined, the file names are `highlight.css` and `highlight.sty`.

NOTE: This module relies on an external program. You thus have to allow `deplate` to run programs, e.g. by using the `-X` command line switch.

8.2.5 `endnotes`: Turn footnotes into endnotes

This module turns footnotes into endnotes that can be listed using the command:

```
#LIST: endnotes
```

The module uses the “endnotes” package in LaTeX format.

8.2.6 `entities-decode`: Insert characters as entities

This module makes it possible to insert characters as entities.

Example 8.1: Entities

```
#VAR: encoding=utf8
```

```
alpha &#945; and beta &beta;.
```

The entities are stored as tab-separated tables (3 columns: printable character, named, numbered) in:

- `ents/NAME-FORMATTER_ENCODING.entities`
- `ents/NAME-ENCODING.entities`

NAME is given by the `entities` variable. If undefined, use `general`. Currently, only UTF-8 encoded entities files are provided.

8.2.7 `entities-encode`: Insert characters as entities

The opposite of `entities-decode`. Encode special characters.

8.2.8 iconv: Convert text between encodings

The source encoding is defined in the `encoding` variable (default: “latin1”), the target encoding in the `recodeEncoding` variable (default: “utf-8”). So, your input text file can be in latin-1 and your docbook output in utf-8.

8.2.9 imgurl: Insert urls/links to images as images

Insert urls/links to images as images and not as references.

8.2.10 linkmap: Define labels refering to URLs

Jeff Barczewski took the idea of markdown to not include URLs in the text but labels that are replaced with the URLs and adapted it for deplate.

This module also modifies the `ref` macro (see 12.2) to use these labels.

Example:

Syntax for using links in map (any of following):

- `[[Example]]`
- `[[Example]][This example]`
- `[[Example][Example in a box]*]`, i.e. in a new window
- `[[Example]$]` no follow rel

or to embed raw URL using `macro{ref: Example} foo`.

Use the `ref` macro but display the URL `{ref p!: Example}`.

Use the `ref` macro but display a different name `{ref name=this example: Example}`.

```
#LinkMap <<---  
Example: http://www.example.com/foo.php&bar=123#abc  
---
```

yields:

Syntax for using links in map (any of following):

- Example
- This example
- Example in a box, i.e. in a new window
- Example no follow rel

or to embed raw URL using `macro Example foo`.

Use the `ref` macro but display the URL `http://www.example.com/foo.php&bar=123#abc`.

Use the `ref` macro but display a different name `this example`.

NOTE: The `LinkMap` environment can be located anywhere in the current source. Labels are effective only in the current source unless `global!` is set.

8.2.11 makefile: Create Makefiles

This module creates a Makefile and exits from the current run. The Makefile should be suitable for most tasks.

Example 8.2: The makefile module

First run:

```
deplate -m makefile -m de -m html-obfuscate-email index.txt
```

This will create Makefile and Makefile.config. Then, you can run, e.g.,

```
make website
```

to create a multi-page website (HTML output, using the htmsite formatter).

NOTE: In case you want to create new make goals, put them into Makefile.config. If you run the above command again, the Makefile will be overwritten but not Makefile.config.

The following goals might be of use:

cleantex Remove temporary LaTeX files

dbk Create docbook output

dvi Create dvi output (via LaTeX)

dviclean Create dvi output and remove temporary files

html Create single-page HTML output (the default)

man Create a man page (via DocBook & `xm1to`)

pdf Create PDF output (via LaTeX)

pdfclean Create PDF output and remove temporary files

php Create PHP output

tex Create LaTeX output

text Create plain text output

website Create multi-page HTML output

The generated Makefile supports implicit, suffix-based rules. Example:

```
make extra.html
```

This will convert the file `extra.txt` (because in the above example the suffix was `txt`) and convert it to single-page HTML, using the command line options defined when creating the makefile.

8.2.12 mark-external-urls: Mark external references with icons

URLs will be marked with images – “mailto.png” or “url.png” depending on the type. The image names can be redefined by setting the document options “mailtoIcon” or “urlIcon”. In this manual I use icons from the QBullets set.

8.2.13 numpara: Numbered paragraphs

This module adds a running number to each formatted paragraph. The paragraphs are numbered as they are formatted, which means that clipped paragraphs, non-standard document slots, or paragraphs in headers or footers are likely to result in a “non-linear” sequence. Add the attribute/option `noNum` to these “out-of-order” paragraphs.

Example 8.3: The numpara module: Don’t number specific buffers

```
#Clip id=clippedParagraph <<--
Some text that forms a paragraph.
#OPT: noNum!
--
```

If you want to change the way paragraphs are numbered redefine the method `Deplate::Element::Paragraph#add_number` like in this example:

Example 8.4: The numpara module: Change numbering style

```
class Deplate::Formatter
  def numbered_paragraph(text, number)
    return [plain_text("§"), number, " ", text].join
  end
end
```

8.2.14 particle-math: LaTeX-like $\$ \backslash math \$$ markup

This module provides a particle that enables you to markup mathematical text by enclosing it in dollar signs as it is used in LaTeX. This is not enabled by default in order to reduce the number of special characters. The text is converted as if you had used the `math` macro (see 12.5).

8.2.15 smart-dash: Insert long or short dashes depending on the context

This module defines a single dash (“-”) as a symbol and turns it into a long dash (i.e. “—”) in some situations:

- when preceding character is a digit
- when the next character is a digit
- when the preceding and the next character are wiki word characters

This module also removes the double dash `--` (see 9.10) from the symbol table. You can force long dashes by using `'--`.

8.2.16 smiley: Replace text smileys with images

Currently only the basic smiley is being recognized.

:-) 😞

You can define your own smileys by adding something like this to `~/deplate/after/mod/smiley.rb`:

```
Deplate::Particle::Smiley.def_smiley(':-(', 'smiley_sad')
```

The suffix of the image file is controlled by the `smileySfx` or `imgSfx` variables.

8.2.17 utf8: Improve unicode awareness

When using this module, `deplate` takes care of UTF-8 multibyte sequences when sorting the index and similar occasions. This module also sets the output encoding to “UTF-8”.

8.2.18 xmlrpc: Work as a xmlrpc server

This modules makes `deplate` work as an xmlrpc server. The xmlrpc server could run in an protected context and make `deplate` formatting capabilities available in an unsafe environment.

This module is experimental and not well tested. The module takes no precautions with respect to concurrent formatting requests.

The xmlrpc server knows the following handlers/methods:

`convert(format, text)` return the converted string

`convert_string(format, text)` same as above

`convert_file(format, filename)` convert the file `FILENAME` and return the name of the output file (this requires the server and the client to work on a shared directory)

`string_to_fileset(format, filename, text)` convert text and return a hash (or whatever) that contains filenames and their contents; this hash comprises auxiliary files too; it's up to the client to save them in a proper place; cross-references assume that these files reside in the same directory

`fileset_to_fileset(format, main_filename, fileset_as_hash)` convert a fileset (a hash of names and contents) to a fileset

NOTE: The xmlrpc server creates a new converter for each request unless the `xmlrpcReuseInterpreter` variable ist defined, which will make it reuse the previous converter (the consequence of which is that abbreviations, indexes and the like from earlier documents are still active). This was the default behaviour before version 0.7.3.

NOTE: This module isn't available in the win32-exe distribution. You have to use the source distribution or the ruby gem.

Example 8.5: Invoke as a xmlrpc server

The server (we must give a file name as argument so that `deplate` doesn't complain about missing arguments):

```
a> deplate -m xmlrpc -
[2005-01-28 20:02:15] INFO WEBrick 1.3.1
[2005-01-28 20:02:15] INFO ruby 1.8.1 (2003-12-25) [i386-cygwin]
[2005-01-28 20:02:15] INFO WEBrick::HTTPServer#start: pid=2536 port=2000
[...]
```

A client:

```
b> irb
irb(main):001:0> require 'xmlrpc/client'
irb(main):002:0> deplate = XMLRPC::Client.new("localhost", "/deplate", 2000)
irb(main):003:0> puts deplate.call("convert", "html", "Some text ...")
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="en">
<head>
[...]
```

```

irb(main):004:0> puts deplate.call("convert", "latex", "Some text ...")
\documentclass[11pt,a4paper,english]{article}
\usepackage[latin1]{inputenc}
\usepackage[]{suppl}
[...]

```

There are some variables to change the port and the path:

xmlrpcAllow a space separated list of IP addresses or regular expressions matching valid IP addresses

xmlrpcPort default is 2000

xmlrpcPath default is “/deplate”

8.3 Syntax

8.3.1 syntax-region-alt: Alternative syntax for regions

This module provides an alternative Syntax for regions.

Example 8.6: The syntax-region-alt module

```

===== Region options: args
content
===== Optional Comment

```

There have to be at least 4 equal signs at the beginning of the line. The number of equal signs in the end statement has to match the number in the begin statement.

8.3.2 markup-1: Re-enable pre0.6 text styles

Since 0.6 the markup for textstyles was changed in order to choose characters that are less likely to be appearing in ordinary text for markup. Use this module to re-enable the old pre 0.6 markup:

Emphasize `*word*`, `**any text**`

Typewriter, Code `=word=`, `==any text==`

8.4 LaTeX

8.4.1 inlatex-compound: Compile LaTeX bits as one file

This (still experimental) module changes the way `deplate` compiles LaTeX bits (e.g., when using the `ltx` or `math` macros). Usually `deplate` would save each bit into one file and run this file through LaTeX. This approach is simple to manage but has the disadvantage that LaTeX commands defined in one bit are unknown in another.

This module makes `deplate` collect all LaTeX bits in one file. Afterwards, the resulting postscript file is split in pieces in order to create one image per page.

If a LaTeX bit covers more than one page, this module fails.

8.4.2 koma: Support for the Koma-Script class

Some support for koma script's document classes:

- Use `scrartcl` instead of `article`
- Use `scrpage2` for headers and footers

8.4.3 latex-emph-table-head: Emphasize head rows

Emphasize header cells and add a ruler after a header row.

8.4.4 latex-verbatim-small: Print verbatim regions in small font size

In order to make it easier for LaTeX, this module prints verbatim regions in small font size.

8.4.5 latex-styles: Styled LaTeX output

This module provides the infrastructure for styling LaTeX output. Currently, only a few table styles are supported. Please see 9.7 for examples.

8.5 HTML

8.5.1 soffice: Support for Star/Sun/OpenOffice HTML

Not much yet:

- Footnotes, headers, and footers are marked in OpenOffice compatible format by default
- Make the `pagenumber` macro insert a reference to the `PAGE` field

8.5.2 html-asciimath: Support for ASCIIMathML.js

ASCIIMathML.js by Peter Jipsen translates plain text formulas to MathML. As it happens to know some LaTeX syntax, it can be used to translate the `math` macro (p. 81) to MathML.

`ASCIIMathML.js` does the translation when viewing the document in your browser which could result in some delay when viewing large files. ASCIIMathML requires JavaScript to be enabled and works with the following browser:

- Internet Explorer 6 + MathPlayer
- Netscape7.1/Mozilla/Firefox

If you don't use this module, the LaTeX code will be embedded as image.

Don't forget to copy `ASCIIMathML.js` to the output directory.

8.5.3 html-headings-navbar: Insert a navigation bar before level 1-headings

This module defines a style which insert a simple navigation bar with a drop-down menu and forward/backward buttons before every level-1 heading.

You can set `headingsNavbarMaxLevel` (default=1) in order to add a navbar for deeper nested headings too.

8.5.4 `html-jsmath`: Support for `jsMath.js`

`jsMath.js` serves a similar purpose as `ASCIIMathML` (p. 39).

8.6 `html-mathml`: Support for the `mathml` gem

To make this work, you have to

1. install `mathml` by KURODA Hiraku
2. use `xhtml11m` (p. 28) as output format.

8.6.1 `html-obfuscate-email`: Obfuscate e-mail

This module replaces e-mail addresses (but only those that have a `mailto` prefix) with some presumably harvester-safe representation. The JavaScript representation is an array of hex chars, which is a method used by several other tools. The no-script representation has the `mailto` prefix removed and the characters `@` and `.` are replaced with `AT` and `DOT`.

8.6.2 `html-sidebar`: Display a sidebar containing a small table of contents

If JavaScript is enabled, you can move the mouse to the left side to view a small table of contents.

8.6.3 `navbar-png`: Display images in the navigation bar

This module redefines the buttons in the navigation bar for using images: `prev.png`, `home.png`, `next.png`

8.7 HTML Slides

8.7.1 `htmlslides-navbar-fh`: Modified navigation bar

This module, which was contributed by Fritz Heinrichmeyer, provides a modified navigation bar using span tags instead of a table and is meant as a replacement for the `html-website` formatter.

If you want this navigation bar to be the default, you can either require it in your `config.rb` file or copy/move/link it to `~/deplate/after/fmt/htmlslides/`.

8.8 Docbook

8.8.1 `symbols-*`: Modify the representation of symbols

If you define the document option “`sgml`”, then the formatter won’t insert symbols in plain utf-8 but in something programs like `jade` deal with more docile.

When writing man pages, you might want to include the module “`symbols-plain`” instead.

8.8.2 `noindent`: Avoid insertion of spaces and newlines

If you define the document option “`sgml`”, the formatter will also load this module which avoids inserting spaces and newlines. This makes programs like `jade` output more nicely formatted documents.

8.9 Miscellaneous

8.9.1 DeplateString

The file `deplate/deplate-string` provides the class `DeplateString`, which is a subclass of `String` and has methods like `#to_html`, `#to_latex` etc. for easier use of this whole thing from within another program.

```
puts DeplateString.new('bar __foo__ bar').to_html
```

8.9.2 Nukumi2

There is a somewhat experimental support to use `deplate` as markup language for a Nukumi2 maintained site. In order to enable `deplate` for Nukumi2, you have to add this line to your Nukumi2 `config.rb`:

```
require 'deplate/nukumi2'
```

and maybe also, in order to enable inline LaTeX, Ruby code etc.:

```
DeplateString.deplate_options.allow += ['x', 'X']
```

A blog entry could then look like this:

```
#TITLE: Deplate Test
#DATE
#KEYWORDS: deplate
```

```
* It works!
```

```
Yes, this article was converted using ''deplate''.
```

This hack was written for Nukumi2 v0.5. It's not guaranteed to work with other versions.

NOTE: There must be an empty line after the header section.

9 Markup

The markup corresponds to the markup of Vim's `wiki-plugin`. There are some minor deviations, though.

This chapter is mainly a test course.

9.1 The Basics

9.1.1 Elements and Particles

A `deplate` document is made up of elements (a.k.a. block elements in HTML, XML etc.) that consist of particles (a.k.a. inline elements). No particle can cross element boundaries. This distinction is important to understand, as commands (p. 70) and regions (p. 54) yield elements, but macros (p. 77) yield particles.

Although `deplate` has some facilities for program control (conditional or repeated output), it really is rather a markup language but no programming language. Although you can define new elements right within a `deplate` document using `deplate` markup, this only works for really simple elements.

If a line matches an element pattern (usually defined by the beginning of the line), it marks the end of the previous element and starts a new element. If a line matches no pattern, it either starts a new paragraph or is added to the previous element. Elements like headings, anchors, or table rows are one-line patterns to which no unmatched line will be added. Other elements like list items or paragraphs can be made up of more than one line.

9.1.2 Backslashes

The `deplate` markup is based on inserting special characters into the text as in this example: `__emphasized__`, which will be printed as *emphasized*. If you want to prevent `deplate` from interpreting these markers, they must be preceded by a backslash.

- The backslash is used as escape character that prevents the wikification of the following character.
- There are two exceptions:
 1. A backslash at the end of the line makes a pattern include the next line; any whitespace at the beginning of the following line is removed.
 2. A backslash in front of a blank results in non-breaking space.
- A backslash that should be printed must be preceded by a backslash.

9.1.3 Special Characters

You should especially take care with the following characters:

Curly braces ({}) Curly braces usually enclose macros; in other contexts, they should always be preceded by a backslash

- There are some grey areas though that could cause problems, e.g. when input to a macro (e.g. the ruby macro) contains an unmatched curly brace

Backticks (`) Backticks are used to insert some symbols and should be escaped by a backslash in other contexts than inserting a special character.

Sequences of character you should notice:

Two underscores (__) followed by non-whitespace Emphasized text

Two single quotes (") followed by non-whitespace Literal text set in typewriter font

Two colons (::) surrounded by whitespace Used in definition lists

Characters with special meaning if they occur at the beginning of line:

Asterisks (*) Headings, unordered lists

Sharp sign (#) Anchors, commands, regions, ordered lists

Percent (%) Comments

List markers 1., a., -, *, #, @, ?, ?.

Other:

The dollar sign (\$) Although it has currently to be enabled by loading a module, there is a chance that this will become the standard delimiter for mathematical expressions typed in latex (with the AMS packages enabled). The standard markup will be either $\$a = x + y\$$ or with blanks $\$ a = x + y \$$ as the dollar sign is most likely used before or after a digit.

Be aware that your editors line wrap can inadvertently place these characters at the beginning of a line.

Example 9.1: Text styles, backslash

```
__emphasize__, WikiName
\__noemphasize\__, \NoWikiName
One \
\
line
One\
  word
unnnnnnnnnnnnnnnnnn\ breakkkkkkkkkkkkkkkkk\ ableeeeeeeeeeeeeeeeeeeee
```

yields:

```
emphasize, WikiName
__noemphasize__, NoWikiName
One line
Oneword
unnnnnnnnnnnnnnnnnn breakkkkkkkkkkkkkkkkkk ableeeeeeeeeeeeeeeeeeeee
```

9.1.4 Argument Values

Arguments to macros, commands, and regions:

- In argument values, the characters !=: and single double quotes must be preceded by a backslash
- Alternatively, you can enclose a value in double quotes (which will be stripped off; double-quotes must be escaped using a backslash) or parentheses (which will be retained)
- Argument values are stripped of whitespace; if you want an argument to contain leading or trailing whitespace, you have to enclose the argument in double quotes

There are two special arguments (`fmt` and `if`) that allow some control on whether an element will be included in the output. Please see 14.6 for details.

Table 3: Key arguments

Example	Key Arguments	Body
<code>{macro}</code>	<code>{}</code>	<code>““</code>
<code>{macro: foo}</code>	<code>{}</code>	<code>“foo“</code>
<code>{macro boo!: foo}</code>	<code>{“boo“ ⇒ true}</code>	<code>“foo“</code>
<code>{macro boo! bar=1: foo}</code>	<code>{“boo“ ⇒ true, “bar“ ⇒ “1“}</code>	<code>“foo“</code>
<code>{macro bar=object(id=1): foo}</code>	<code>{“bar“ ⇒ “object(id=1)“}</code>	<code>“foo“</code>
<code>{macro bar=“foo := bar“: foo}</code>	<code>{“bar“ ⇒ “foo := bar“}</code>	<code>“foo“</code>
<code>{macro bar=(boo=boo): foo}</code>	<code>{“bar“ ⇒ “(boo = boo)“}</code>	<code>“foo“</code>
<code>{macro bar=\(boo=boo): foo}</code>	<code>{“bar“ ⇒ “(, “boo“ ⇒ “boo)“}</code>	<code>“foo“</code>
<code>{macro bar=“(boo=boo): foo}</code>	<code>{“bar“ ⇒ “(, “boo“ ⇒ “boo)“}</code>	<code>“foo“</code>

9.2 Comments (whole lines)

- The percent sign has to be first non-blank character in the line. Otherwise it’s interpreted as a character.
- A comment separates lines as paragraphs.
- For intra-paragraph comments use the comment macro 12.2

Example 9.2: Comments

```
Line %1
%Comment
Line %2
```

yields:

```
Line %1 Line %2
```

9.3 Paragraphs

A paragraph a sequence of non-empty lines that don’t match anything else.

Example 9.3: Paragraphs

```
Paragraph 1 finishes with an empty line. Paragraph 1 finishes with an
empty line. Paragraph 1 finishes with an empty line.
```

```
Paragraph 2 finishes with an unordered list. Paragraph 2 finishes with
an unordered list. Paragraph 2 finishes with an unordered list.
```

- ```
- Item 1
- Item 2
```

yields:

```
Paragraph 1 finishes with an empty line. Paragraph 1 finishes with an empty line.
Paragraph 1 finishes with an empty line.
```

```
Paragraph 2 finishes with an unordered list. Paragraph 2 finishes with an unordered
list. Paragraph 2 finishes with an unordered list.
```

- Item 1
- Item 2

## 9.4 Headings

*NOTE:* Headings can span more than one line by putting a backslash ('\') at the end of the line.

Known options:

**noList!** prevent the heading from being listed in the toc; as you can see from the current heading, the heading number is increased nevertheless (which probably should be considered as a bug?)

**caption** the display name in the table of contents

**shortcaption** the display name in a minitoc; with multi-file output, this argument defines the optional file name

**id** similar to shortcaption; if an id and a shortcaption are given, the id will be used for the filename and the sort caption for the mini-toc

**plain!** Don't add any numbering to the heading. (Set the variable `headings` to "plain" if you want to turn off numbering for all headings.)

**url=URL** Turn a heading into a hyperlink; this requires the `hyperHeading` variable(see 14.1.2) to be set

If you set the variable `autoFileNames`, `deplate` will derive the file name from the heading.

```
* Level 1
#OPT: id=foo

** Level 2

*** Level 3
#OPT: noList! plain!
...
```

## 9.5 Lists (indented)

### Example 9.4: List

```
- Item
 * Item
 + Item
 1. Item 1
 a. Item a
 B. Item B
 #Verb <<-----
 EMBEDDED VERBATIM TEXT

 2. Item 1
 || Embedded || Table ||
 | x | 1 |
 | y | 2 |
 * Item
 # Item 1
 © There is much to say about Item A, too much to be put
 in words or to be written down.

 What else is there to say about Item A?
```

```

 @ Item B
 # Item 2
- Do this
 #A _ Some task
 #B1 _ Some other task
- Do that
 #A x11-11-2050 Something was done!
 #B1 11-11-2050 Some other task

```

yields:

- Item
  - Item
    - \* Item
      1. Item 1
        - a. Item a
        - B. Item B
 

EMBEDDED VERBATIM TEXT
      2. Item 1
 

|          |       |   |
|----------|-------|---|
| Embedded | Table |   |
| x        |       | 1 |
| y        |       | 2 |
  - Item
    1. Item 1
      - a. There is much to say about Item A, too much to be put in words or to be written down.  
What else is there to say about Item A?
      - b. Item B
    2. Item 2
- Do this
  - A** Some task
  - B1** Some other task
- Do that
  - A (11-11-2050)** Something was done!
  - B1 (11-11-2050)** Some other task

*NOTE:* “#” (numbered, ordered lists), and “@” (ordered lists with letters) are the preferred markers. If you use dashes as in LaTeX (i.e. “–”), a dash appears to be okay too. Be aware that, like LaTeX, `deplate` concatenates two normal dashes (“–”) to one long dash (“—”) – see also 9.10.

*NOTE:* If you run into troubles when nesting lists, consider the use of the list macro 12.5.

*NOTE:* Task lists are implemented as `varlist` in DocBook.

## 9.6 Description lists (indented)

### Example 9.5: Description list

```
Item :: Description
```

yields:

**Item** Description

## 9.7 Tables

A table is marked as sequence of lines beginning and ending with one or two “|” symbols. Two pipe symbols (or whatever its name is) define a row belonging to the head or the foot.

A row containing only empty cells and cells with hyphens defines a ruler.

A cell containing only “^” means concatenate with the cell above. A cell containing only “<” means concatenate with the cell to the left.

Tables take the following options (via the OPT command):

`head=N` The number of rows encompassing the table head

`foot=N` The number of rows encompassing the table foot

`hiRow=N1,N2,first,last,...` The rows to be highlighted

`hiCol=N1,N2,first,last,...` The columns to be highlighted

`rows=ROWDEF1, ROWDEF2, ...` Define a row’s attributes

- ROWDEF has the form: `KEY1.VAL1 KEY2.VAL2 ...`; KEY being one of
  - h** The row height
- a ruler counts as a row

`cols=COLDEF1, COLDEF2, ...` Define a columns’s attributes

- COLDEF has the form: `KEY1.VAL1 KEY2.VAL2 ...`; KEY being one of
  - w** The column width
  - j** The column’s justification (left, right, center, or justified)
  - r** Add a left-handed vertical ruler (the value denotes the number of rulers or its thickness)
    - This option is currently only interpreted for LaTeX output

`long!`, `short!` Define whether the `longtable` environment should be used in LaTeX output (by default, the `longtable` environment will be used for tables with more than 20 rows).

- Switching between the `table` and the `longtable` environment has some effects on the layout. One consequence is that footnotes are set properly only in the `longtable` environment. On the other hand, the `here`, `floatHere`, `align`, and `floatAlign` options are currently ignored.

`style=NAME` The table style (but see below)

`stylex=NAME` Like `style` but ignore `tableStyle`

`note=TEXT` A short note concerning data source etc.

In addition, the following options for “floats” apply too:

`here!`, `floatHere=1` In LaTeX output, add the `h` attribute to the table environment

`align=[left|right|center]`, `floatAlign=[left|right|center]` align the table; the first one is a table specific options, the second a global document option

**Table 4:** This Table

| Head | Categories |
|------|------------|
| Row1 | Value1     |
| Row2 |            |
| Row3 | Value3     |

Source: My Head, 2004

*Joining cells:* If a cell contains nothing but “^” or “<”, the current cell will be joined with the left one or with the cell above. (The actual output depends on the formatter and the capabilities of the target format, though.)

It depends on the formatter whether these options are interpreted.

**Example 9.6: Table**

```

	Head		Categories		<	
Row1	Value1	Value1				

Row2		Value2				

| Row3 | Value3 | ^ |
| ----- |
#OPT: hiCol=first hiRow=last cols=w.3cm j.right r.1, w.1cm,,r.1 rows=h.3cm
#OPT: note=Source: My Head, 2004
#CAP: This Table
#thisTable

```

yields:

*NOTE:* Rows can span more than one line by putting a backslash (‘\’) at the end of the line.

*NOTE:* An empty cell is marked by at least 3 consecutive blanks.

You can also define a style for a table. The style attribute is currently only interpreted by the latex-styles module and to some degree emulated in HTML output – not so surprisingly, this works better in Firefox than in MS IExplorer.

The following styles are known (for one or another output format):

**grid** Draw all table borders/rulers

**formal** Draw a ruler at the top and the bottom of the table, as well as between groups

**box** Like formal but add vertical rulers on both sides

**overlay** Like box but draw the background of some rows and columns in gray or in colours

**list** Horizontal rulers between each row

**small, footnotesize, scriptsize** Smaller font sizes

**dense08** decreased intercell spacing (there is a minor problem with line spacing)

**landscape** rotate the table (LaTeX only, I assume)

**Example 9.7: Table styles**



```

#Var id=styledTableExample <<---
	Head		A		B	
Row1	1	2				
Row2	3	4				
	Foot		X		Y	
#OPT: hiCol=first

Test ''grid'' & ''scriptsize'' styles:
#INCLUDE var=styledTableExample
#OPT: style=grid,scriptsize

Test ''formal'' & ''footnotesize'' styles:
#INCLUDE var=styledTableExample
#OPT: style=formal,footnotesize

Test ''box'' & ''small'' styles:
#INCLUDE var=styledTableExample
#OPT: style=box,small

Test ''overlay'' style:
#INCLUDE var=styledTableExample
#OPT: style=overlay,landscape

```

yields:

Test grid & scriptsize styles:

|      |   |   |
|------|---|---|
| Head | A | B |
| Row1 | 1 | 2 |
| Row2 | 3 | 4 |
| Foot | X | Y |

Test formal & footnotesize styles:

|      |   |   |
|------|---|---|
| Head | A | B |
| Row1 | 1 | 2 |
| Row2 | 3 | 4 |
| Foot | X | Y |

Test box & small styles:

|      |   |   |
|------|---|---|
| Head | A | B |
| Row1 | 1 | 2 |
| Row2 | 3 | 4 |
| Foot | X | Y |

Test overlay style:

| Head | A | B |
|------|---|---|
| Row1 | 1 | 2 |
| Row2 | 3 | 4 |
| Foot | X | Y |

## 9.8 Anchors

Anchor (or labels respectively) are attached to the previous element, save if the previous element is a whitespace. In this case the anchor will be attached to the following element.

The name must begin with a lower letter. There mustn't be non-whitespace character before the sharp sign.

### Example 9.8: Anchors

```
* This is section one
#labSectOne

Some text.
#labSomeText

#labSectTwo
* This is section two
```

## 9.9 Wiki Names, URLs

`deplate` is inspired by several wiki engines. It thus also provides formatting of wiki names and URLs as hyperlinks. There are about four types of wiki names.

**Simple** Any word in CamelCase is turned into a wiki name – save if wikification is prevented with a backslash – see 9.1.2.

**Quoted** Any text between `[-` and `-]` will be turned into a hyperlink.

**Extended** An extended wiki name has the form: `[[DESTINATION] [OPTIONAL NAME]MODIFIER]`.

- MODIFIER is optional.
  - “\*” open page in new window
  - “\$” set `rel=‘nofollow‘`
  - “!” prevents `deplate` from adapting the reference's suffix
    - `deplate` was created as the publishing framework (or so) for a personal wiki (namely the Vim wiki plugin). As such it assumes that the files being referenced to are converted to the same target format as the current file, which is why `deplate` modifies an extended wiki destination's suffix. Use this modifier to prevent `deplate` from doing so.
    - Often the chosen `deplate` output is only an intermediary file; in order to make `deplate` append the suffix of the final output file, set the document option `suffix` to the desired value. E.g. if you convert to docbook and then from docbook to html, pass the option `-D suffix=html` to `deplate` when converting the text sources.
- If you want a hyperlink to have an image, use:  
`[[http://www.example.com][{img: example_image}]]`. See also 12.5.

**InterWikis** An interwiki name is a shortcut to a different wiki. Any simple and quoted wiki name can be deferred to an interwiki by prepending `NAMEINCAPITALS::`. You have to define an interwiki before referring to it by adding something like this to your configuration file: `InterWiki.add('ID', 'http://BASEURL/', '.SUFFIX')`

Wiki Names are automatically marked as index entries – see 11.4 and 11.5.

### Example 9.9: Wiki names

```
WikiName
WikiName#anchor

[-name-]
[-some name-]#there
[--]#here

OTHERWIKI::WikiName
OTHERWIKI::WikiName#there
OTHERWIKI::[-some name-]
OTHERWIKI::[-some name-]#there

[[destination]]
[[destination] [name]]
[[destination#anchor] [name]]
[[destination.suffix]!]
[[OTHERWIKI::destination#anchor] [name]]
[[#anchor]]
[[#anchor] [name]]
```

## 9.10 Symbols

The backtick (`) is used to introduce some symbols like quotation marks (e.g., in positions where `deplate` would choose the wrong one).

Symbols are not expanded in text marked as typewriter/code and in verbatim regions.

### Example 9.10: Symbols

```
<-, ->, <=, =>, <~, ~>, <->, <=>, <~>, !=, ~~, ..., --, ==, ‘‘, ’’, ‘, ’
’’->’’, ’’<-’’

#Verb <<--
<-, ->, <=, =>
--

#Code id=symbolsTest syntax=ruby <<--
<-, ->, <=, =>
--
```

yields:

```
←, →, ⇐, ⇒, <~ , ~> , ↔ , ⇔ , ⇌ , ≠ , ≈ , … , − , ≡ , “ , ” , ‘ , ’
->, <-

<-, ->, <=, =>
```

*NOTE:* If you want to disable some of these symbols, you could add some ruby code like this to your `config.rb`:

## 9.11 Markers

Markers are meant to highlight a position in the text. In order to avoid ambiguities with notes (see below), there should not be a space before or after the marker – i.e. it should immediately follow or precede the word it is referring to.

### Example 9.11: Markers

```
- elaborate+++
- here###
- questionable???
- attention!!!
```

yields:

- elaborate\*
- here\*
- questionable\*
- attention\*

```
* +++
* ###
* ???
* !!!
```

## 9.12 Notes

Indented paragraphs that begin with a marker and a subsequent space start an annotation, i.e. the text of paragraph will be turned in a margin note or similar – depending on the output format.

### Example 9.12: Notes

```
Indented paragraphs that begin with a marker and a subsequent space
start an annotation.
```

```
!!! Something like this.
```

yields:

```
Indented paragraphs that begin with a marker and a subsequent space start an anno-
tation.
```

```
Important:
Something li
this.
```

## 9.13 Strings, Smart Quotes

### Example 9.13: Quotes

```
"Text in quotes"
```

yields:

```
“Text in quotes”
```

*NOTE:* Quotes are handled by two classes: `Deplate::Particle::DoubleQuote` and `Deplate::Particle::SingleQuote`. In order to disable smart quotes add this code to your `config.rb`:

## 9.14 Textstyles

### Example 9.14: Text styles

```
__emphasize__, 'typewriter'
```

yields:

```
emphasize, typewriter
```

*NOTE:* There must not be a whitespace after the opening mark.

*NOTE:* Text in typewriter style is basically set verbatim with the exception of backslashes.

## 9.15 Breaks

A break is marked by at least 2 dashes, “8<”, and again at least 2 dashes.

```
I say so.
----8<-----
Summary:
```

In HTML output, a break is represented by a horizontal line. In other output formats, a break actually denotes a page break.

## 9.16 Whitespace

A line of whitespace usually separates text elements – e.g. paragraphs,

# 10 Regions

The text marked by a region is interpreted according to the region’s rules. The text could be normal deplate source, but it could also be ruby code, LaTeX, or whatsoever.

Check for end pattern:

```
#Type [OPTIONS] <<----
Text ...

```

Match until next empty line:

```
#Type [OPTIONS] <<
Text ...
```

Deprecated:

```
#Type [OPTIONS]:
Text ...
#End
```

This form can’t be nested.

“Type” being one of:

- Abstract

- Code
- Define
- Doc, Var
- Footnote, Fn
- For
- Header, Footer
- Img
- Inlatex, Ltx
- Native or the name of the chosen formatter
- Quote
- Set
- R (tables)
- Region (generic)
- Ruby
- Swallow, Skip
- Table
- Verbatim, Verb
- Write

## 10.1 Abstract

### Example 10.1: Abstract

```
#Abstract <<EOA
''Deplate'' is ruby script for converting wiki-like markup to latex,
html, or "html-slides". The markup was originally based on the
emacs-wiki mode.
EOA
```

yields:

Deplate is ruby script for converting wiki-like markup to latex, html, or  
“html-slides”. The markup was originally based on the emacs-wiki mode.

## 10.2 Code

This region should be used for displaying source code. Depending on the your use of the corresponding highlighter modules (see 8.2.4, 8.2.3) and its capabilities, the source code will be highlighted.

Options:

**id** Allow `deplate` to cache the formatted output

**syntax** The code's syntax

**style** The style that should be used for highlighting

### Example 10.2: Code, syntax highlighting

```
#Code id=Foo syntax=ruby <<---
class Foo
 # Return two times x
 def bar(x)
 x * 2
 end
end

```

yields:

```
class Foo
 # Return two times x
 def bar(x)
 x * 2
 end
end
```

*NOTE:* These corresponding highlighter modules rely on external programs. You thus have to allow `deplate` to run programs, e.g. by using the `-X` command line switch.

## 10.3 Define: DefRegion, DefCommand, DefMacro, DefElement, DefRegionN, DefCommandN, DefMacroN

This is still somewhat experimental and doesn't always work right.

These regions provide a `deplate` only way to define simple macros or rather mini templates. So you don't have to know ruby in order to perform macro/template-expansion kind of stuff.

These regions come with several synonyms:

**DefRegion** DefineRegion, Defr

**DefCommand** DefineCommand, Defc

**DefMacro** DefineMacro, Defm

**DefElement** DefineElement, Defe

**DefRegionN** DefineRegionN, Defrn



**DefCommandN** DefineCommandN, Defcn

**DefMacroN** DefineMacroN, Defmn

The anonymous argument will temporarily (during template expansion) assigned to the document option `@body`, which can be accessed by `#VAR` (p. 72), `{var}` (p. 78), `#ARG` (p. 72), or `{arg}` (p. 79). The `doc` command/macro insert the argument as plain text, while the `arg` command/macro parses the argument.

When choosing argument names, be aware that they are passed as document options (see 14.1) and that some of these arguments (e.g., `id`, `fmt`, `noFmt`, `if`) are implicitly evaluated by `deplate`. The unnamed arguments `@note` and `@body` are assigned by `deplate`.

Macro templates are parsed as normal templates first (see 5.5). As a consequence, macro templates may contain `#IF` commands and the like.

```
#Region NAMED_REGION_ARGUMENTS NAMED_TEMPLATE_ARGUMENTS: @note <<-
@body
-
#Region NAMED_REGION_ARGUMENTS NAMED_TEMPLATE_ARGUMENTS <<-
@body
-

#COMMAND NAMED_COMMAND_ARGUMENTS NAMED_TEMPLATE_ARGUMENTS: @body
#COMMAND NAMED_COMMAND_ARGUMENTS NAMED_TEMPLATE_ARGUMENTS
#COMMAND: @body

{macro NAMED_MACRO_ARGUMENTS NAMED_TEMPLATE_ARGUMENTS: @body}
{macro NAMED_MACRO_ARGUMENTS NAMED_TEMPLATE_ARGUMENTS}
{macro: @body}
```

### Example 10.3: Define a new region

```
#DefRegion id=ThingsIHaveTo <<--
Things I have to {arg default=eat: do}:
#XARG: @body
--
#DefMacro id=thing args=@body <<--
__ {xarg: @body} __
--
% The ''args'' argument make these keys mandatory!
#DefCmd id=THING args=name @body <<--
| {thing: {xarg: name}} | {xarg: @body} |
--

#ThingsIHaveTo do=buy <<---
#THING name=Apples: 1kg
#THING name=Peaches: 2kg

```

yields:

Things I have to buy:

*Apples* 1kg  
*Peaches* 2kg

The examples in this document were printed with this `DefRegion`:

```
#DefRegion id=Example <<--
Example:
#WITH arg=@body: Verb

yields{arg default="": note}:

#ARG: @body
--
```

#### Example 10.4: Example region

```
#Example note=" an example" <<---
__Yes__, this is an example of using the #Example region.

```

yields:

#### Example 10.5: Example region

```
__Yes__, this is an example of using the #Example region.
```

yields an example:

```
Yes, this is an example of using the #Example region.
```

This regions displays its body twice: first wrapped in a `#Verb` and then properly parsed and formatted.

#### Example 10.6: Literate programming

This can also be used for some kind of cheap literate programming experience.

```
#DefRegion id=Source @args=file <<--
\#WITH arg=@body: Write append! file={xarg: file}
\#WITH arg=@body: Code id={xarg:id}
--
```

The `#WITH` command is escaped with a backslash so that the template input filter, which pre-processes the body, doesn't complain about an unknown region (as the template input filter doesn't allow `#WITH` commands).

This region could be used as follows:

```
#VAR: codeSyntax=ruby codeStyle=tomacs
```

So, you would really like to define a new class. Now?

```
#Source id=classFooOpen file=foo.rb <<
class Foo
```

And then maybe even add a method?

```
#Source id=methFoo_test file=foo.rb <<
 def test(a)
 p a * 2
 end
```

Well, it's really up to you.

```
#Source id=classFooEnd file=foo.rb <<
end
```

The command

```
deplate --allow=w,W -m code-gvim71 test.txt
```

would then create two files (test.html and foo.rb). In this simple version the generated documentation and source files would reside in the same directory though. Before re-running the command, you'd have to clear the output directory.

### 10.3.1 Elements

Within some limitations, you can also define your own elements.

#### Example 10.7: Define an element

```
#DefElement rx=(\\s+)\\[\\]\\s+(.*) <<--
{arg: 1}+ {arg: 2}
{arg: 1}#OPT: style=todo
--
#DefElement rx=(\\s+)\\[x\\]\\s+(.*) <<--
{arg: 1}- {arg: 2}
{arg: 1}#OPT: style=todo
--
 [] Something I should do
 but haven't done yet.
 [] Something else
[x] Something I did,
 maybe yesterday.
```

yields:

- Something I should do but haven't done yet.
  - Something else
- Something I did, maybe yesterday.

### 10.3.2 Particles

You can also define particles if you really want to.

#### Example 10.8: Define a new particle

```
#DefParticle: \\^(\\w+?)\\^ <<--
{sup: {arg: 1}}
--
Test: x^2^
```

yields:

Test: x<sup>2</sup>

### 10.3.3 Native templates

The \*N variants of these regions define “formatter native” templates.

#### Example 10.9: Insert text literally

```

#IF: fmt=~html
#DefRegionN id=Pairs <<---
<p>{arg: name}</p>
<table>
#ARG: @body
</table>

#DefCommandN id=LISTPAIR <<---
<tr>
{arg: @body}
</tr>

#DefMacroN id=pair args=name @body <<---
<td>{arg: name}</td><td>{arg: @body}</td>

#Pairs name=A list of pairs <<---
#LISTPAIR: {pair name=<test>: <Foo>}
#LISTPAIR: {pair name=<native>: &}
#LISTPAIR: {pair name=<define>: <Bar>}

#ELSE
Sorry, the example is defined for HTML output only.
#ENDIF

```

yields:

Sorry, the example is defined for HTML output only.

## 10.4 Doc, Var

Defining a document variable using the `#Var` can come handy in conjunction with `#For` (p. 60). It's important to understand that a variable defined by the `#Var` region is internally represented as an array (one entry per line). If you simply insert this array into a document, the result might not look like what you expected it to be. Also, if you write your own modules, using the `#Var` region is a convenient way to pass around information.

## 10.5 For

This region provides a simple way of looping through lists.

Arguments:

**id** The doc variables the values should be assigned to.

**sep** The list separator (if the argument is a string)

**var** The document variable, which holds the list (if this isn't defined, the text after the colon will be used)

### Example 10.10: For

```

#Var: Fruits:
1kg, apples
2kg, peaches
#End

```

```
#For id=listentry doc=Fruits <<--
\#For id=kg,fruit sep=, doc=listentry:
I would like to buy \{arg: kg} of \{arg: fruit}.
#End
--
```

yields:

I would like to buy 1kg of apples.

I would like to buy 2kg of peaches.

It is necessary to escape the inner `#For` region and the `arg` macros with backslashes, because the body is passed through the template filter. You can prevent this by using the `noTemplate!` argument or by setting the `legacyFor1` variable.

## 10.6 Header, Footer

The argument to these regions is either a paragraph or a table with up to three columns. The result of the header and footer region depends on the formatter.

**HTML** Insert the argument at the top/bottom of the document

**LaTeX** Ignore footer, add the header using `\markright{}`

**LaTeX + koma module** Use koma's `scrpage2` package to format the header and the footer; with the koma module these regions take some optional arguments

- center, right (only if the text is a paragraph; the default is to put the header/footer in the left column)
- linesep=`\d+`, linesep=`\d+pt`, linesep! (defaults to 0.4pt)
  - or use the abbreviation “sep”

See also notes on the `pagenumber` (p. 81) macro.

## 10.7 `!img`

`deplate` provides support for including graphics generated “on the fly” by other programs.

**Synonyms** Image, Fig, Figure

Options:

**id** the basis for the filenames (RECOMMENDED, default: `deplatxImgAuto%02d`); if you don't provide an id, `deplate` uses an automatically generated one. Anyway, if you make changes to the document, it's possible that the generated graphics get messed up

**sfx** the graphic file's suffix

Argument (after the colon): `PROGRAM_NAME COMMANDLINE_OPTIONS ...`

- `PROGRAM_NAME` is the program name known to `deplate`, currently

- dot
- neato
- R

\* Note: Options for the R devices can be defined as arguments to the region. In the form “DEV\_OPTION” the option is specific to a chosen device (e.g. “png.width=700”). In the form “\_OPTION”, it is applicable to all devices (e.g. “\_pointsize=10”). This is necessary as some devices expect dimensions to be given in inches, other in pixels. Please refer to the R manual to find out which options are applicable.

- COMMANDLINE\_OPTIONS are pasted somewhere into the command call – how this exactly happens depends on the calling method

The region `Img` returns an object of the same type as the command `IMG` (see 11.5). All arguments are passed on to the resulting object.

### Example 10.11: Image created with dot

Using ‘dot’:

```
#Img id=dot_example: dot <---
digraph structs {
 node [shape=record];
 struct1 [shape=record,label="<f0> A|<f1> B|<f2> C"];
 struct2 [shape=record,label="<f0> D|<f1> E"];
 struct3 [shape=record,label="<f0> F|<here> G"];
 struct1:f1 -> struct2:f0;
 struct1:f2 -> struct3:here;
}

```

#OPT fmt=latex: w=6cm  
#CAP: Example based on the dot manual

Using ‘R’:

```
#Img id=r_geyser: R <---
attach(geyser)
truehist(duration, nbins=20, xlim=c(0.5, 6), ymax=1.2)
lines(density(duration, width="nrd"))

```

#OPT fmt=latex: w=6cm  
#CAP: Example plot based on Venables/Ripley{cite y!: mass4}

yields:

Using dot:

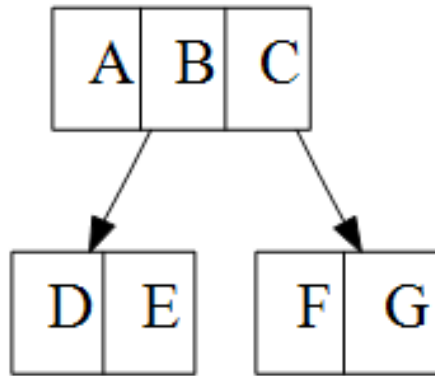
Using R:

## 10.8 Inlatex, Ltx

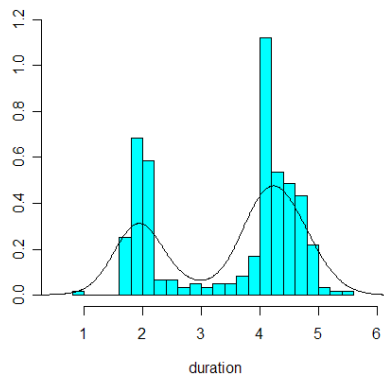
`deplate` contains primitive support for including LaTeX as graphics in non-latex documents. This requires latex, dvips, and ghostscript to be installed.

**Synonyms** `Ltx`

**Figure 1:** Example based on the dot manual



**Figure 2:** Example plot based on Venables/Ripley (2003)



Options:

**id** the basis for the filenames (RECOMMENDED, default: `deplateLtxAuto%02d`); if you don't provide an id, `deplate` uses an automatically generated one. Anyway, if you make changes to the document, it's possible that the generated graphics get messed up

**sfx** the suffix/device for the graphic file (default: `jpeg`)

**rx** the resolution of the graphic file (default: `140`)

**type** “fig” (the default) or “table”

All the options if `#IMG 11.5` apply, too.

By default the following packages are loaded:

- `amsmath`
- `amsfonts`
- `amssymb`
- `mathabx`

If the LaTeX formatter was chosen anyway, the source is inserted literally in the output document. See the `ltx` macro 12.5 for an alternative for inserting smaller pieces of LaTeX-code.

The id is the basis for the filenames being used during conversion. The following files are generated:

- by `deplate`: `FILE.tex`
- by `latex`: `FILE.dvi` + auxiliary files
- by `dvips`: `FILE.ps`
- by `ps2ppm`: `FILE.%02d.SFX`
  - this depends on the `sfx` option

Lines that match `/^\(usepackage|input\)(\[.*\])?\{.\+\}$` are put in the prematter of the latex file, the rest is put in the body.

Lines ending with `%%` are moved to the prematter too.

### Example 10.12: Inline LaTeX

```
#Set: InlatexExampleClip <<EOC
#Inlatex inline! id=InlatexExample sfx=jpeg bh=27 bw=79 <<EOI
\usepackage{graphicx}

$$\sum_{i=1}^n \text{\rotatebox{90}{x}_i}$$

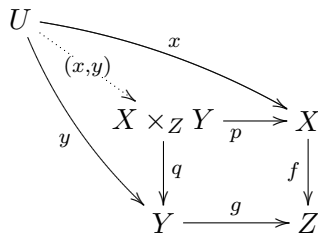
EOI
EOC
```

This is a test (`{get: InlatexExampleClip}`) of a clipped Inlatex region as an inline graphic.

```
#Inlatex id=InlatexExampleXY <<-----
\usepackage[all]{xy}
```



**Figure 3:** An example from the xypic-user guide



```

\parbox{5cm}{
\xyatrix{
U \ar@/_/[ddr]_y \ar@/^/[drr]^x
\ar@{.>}[dr]|-{(x,y)} \\\
& X \times_Z Y \ar[d]^q \ar[r]_p \\
& X \ar[d]_f \\\
& Y \ar[r]^g & Z }
}

#CAP: An example from the xypic-user guide

```

yields:

This is a test ( $\sum_{i=n}^n \xi$ ) of a clipped Inlatex region as an inline graphic.

### 10.9 Native

#### Example 10.13: “Native” text

```

#Native fmt=html <<EON
<p><native> html</p>
EON
#Native fmt=latex <<EON
Pure LaTeX. $\sum_{i=n}^n x_i$
EON

```

yields(depending on the formatter):

Pure LaTeX.  $\sum_{i=n}^n x_i$

### 10.10 Quote

Synonyms: Qu

Options:

**long!** Prepare a long quotation

#### Example 10.14: Quotations

```

#Quote <<---
As I said earlier ...

```

yields:

As I said earlier ...

**Table 5:** A clipped table

clipped table

### 10.11 Set, Get

There are two ways of storing text for later use: clips and variables. Clips should be used for moving around parsed text within the document. Variables should be used for passing text to macros, commands etc.

The `Set` region and the `SET` command can be used to store clips. They differ in that the region results in an array of (block) elements and the command in a sequence of text particles/inline elements. This is important as there are also two corresponding ways to insert clips later on. The `GET` command, which is supposed to insert a block element, and the `get` macro, which works on the inline/text particle level.

| Level   | Region | Command | Macro |
|---------|--------|---------|-------|
| Element | Set    | GET     |       |
| Text    |        | SET     | get   |

#### Example 10.15: Clip

```
#Set: clip <<EOC
| clipped | table |
#CAP: A clipped table
EOC
```

```
#GET: clip
```

```
#SET id=clippedText: clipped text
This is {get: clippedText}.
```

yields:

This is clipped text.

### 10.12 R generated tables

Include tables generated on-the-fly using R. The region takes one optional argument:

**xtable** the result is a html table as formatted with `xtable`; this table won't be used as such but it will be parsed and transformed into a `deplate` table; a `library(xtable)` statement is added to the R source

**verb, verbatim** insert the result verbatim

**drop** the R code doesn't produce any interesting output (data sets, variables and the like should be saved in `.Rdata`)

The default is to generate a pseudo table from the verbatim R output.

Options:

**skip=HEAD skip=HEAD, TAIL skip=, TAIL** Remove the first HEAD lines and the last TAIL lines from the R output

**guess! scanTable!** Try to guess column borders

**Table 6:** An example from the xtable manual using the tli data set

|           | Df | Sum Sq   | Mean Sq | F value | Pr(> F) |
|-----------|----|----------|---------|---------|---------|
| sex       | 1  | 75.37    | 75.37   | 0.38    | 0.5417  |
| ethnicity | 3  | 2572.15  | 857.38  | 4.27    | 0.0072  |
| grade     | 1  | 36.31    | 36.31   | 0.18    | 0.6717  |
| disadv    | 1  | 59.30    | 59.30   | 0.30    | 0.5882  |
| Residuals | 93 | 18682.87 | 200.89  |         |         |

**Table 7:** Summary of the tli data set

| grade        | sex  | disadv | ethnicity   | tlimth       |
|--------------|------|--------|-------------|--------------|
| Min. :3.00   | F:51 | NO :65 | BLACK :23   | Min. :17.0   |
| 1st Qu.:4.00 | M:49 | YES:35 | HISPANIC:20 | 1st Qu.:69.0 |
| Median :6.00 |      |        | OTHER : 2   | Median :80.0 |
| Mean :5.56   |      |        | WHITE :55   | Mean :76.4   |
| 3rd Qu.:7.00 |      |        |             | 3rd Qu.:87.0 |
| Max. :8.00   |      |        |             | Max. :93.0   |

### Example 10.16: Tables created with R

```
#R: drop <<---
data(tli)
x <- tli
save.image(compress=T)

#R: xtable <<---
fm1 <- aov(tlimth ~ sex + ethnicity + grade + disadv, data=x)
fm1.table <- xtable(fm1)
print(fm1.table,type="html")

#CAP: An example from the xtable manual using the tli data set

#R <<---
summary(x)

#CAP: Summary of the tli data set

#R guess! <<---
summary(x)

#CAP: Summary of the tli data set (guessing cell borders)

The same inserted verbatim:
#R: verb <<---
summary(x)

```

yields:

The same inserted verbatim:

```
 grade sex disadv ethnicity tlimth
Min. :3.00 F:51 NO :65 BLACK :23 Min. :17.0
```

**Table 8:** Summary of the tli data set (guessing cell borders)

| grade        | sex  | disadv | ethnicity   | tlmth        |
|--------------|------|--------|-------------|--------------|
| Min. :3.00   | F:51 | NO :65 | BLACK :23   | Min. :17.0   |
| 1st Qu.:4.00 | M:49 | YES:35 | HISPANIC:20 | 1st Qu.:69.0 |
| Median :6.00 |      |        | OTHER : 2   | Median :80.0 |
| Mean :5.56   |      |        | WHITE :55   | Mean :76.4   |
| 3rd Qu.:7.00 |      |        |             | 3rd Qu.:87.0 |
| Max. :8.00   |      |        |             | Max. :93.0   |

|              |      |        |             |              |
|--------------|------|--------|-------------|--------------|
| 1st Qu.:4.00 | M:49 | YES:35 | HISPANIC:20 | 1st Qu.:69.0 |
| Median :6.00 |      |        | OTHER : 2   | Median :80.0 |
| Mean :5.56   |      |        | WHITE :55   | Mean :76.4   |
| 3rd Qu.:7.00 |      |        |             | 3rd Qu.:87.0 |
| Max. :8.00   |      |        |             | Max. :93.0   |

### 10.13 Region

This is a generic region that can be used, e.g., to attach a style to some blocks. For HTML output, this would wrap the blocks in a div and define a class attribute.

### 10.14 Ruby

If a ruby region returns an array, its elements are treated as lines that are joined with “\n”.

Options:

**plain!** reformat ruby’s output as a plain text paragraph (default)

**verb!** put ruby’s output in a verbatim region

**image=FILENAME** the result is a FILENAME to be included

**native!** if neither the verb! nor the plain! option are given, consider the output to be already formatted.

**context=ruby|deplate|self** sets the context and the method that are used to evaluate the code

**ruby** use eval

**deplate** use the current instance of `Deplate::Core`

**self** use the invoking instance

The ruby code is currently simply evaluated in the context of the `Deplate::Core.eval_ruby` method. Inline ruby code can thus access a variable “caller” that refers to the instance, which invoked this evaluation. In future releases, the evaluation of ruby code will take place in a secured thread.

#### Example 10.17: Inline ruby

```
#Ruby context=deplate <<---
def a(x)
 return x*2
end
```

**Table 9:** An example of an char separated table

| Foo | Bar |
|-----|-----|
| 1   | 2   |

```
"Evaluated for #{@formatter.formatter_name}:
Ruby output: #{a(3)}"

```

yields:

Evaluated for latex: Ruby output: 6

### 10.15 Swallow, Skip

Swallow/Skip (= don't display) the text in this region. Process it nevertheless for possible side-effect – i.e., variables being set etc.

### 10.16 Table

This region takes some kind of text and transforms it into a table. Currently, only character/string separated tables are supported

Options:

**sep** the cell separator regexp (default: “\t”)

#### Example 10.18: Create a table from tab-separated input

```
#Table sep=\\. caption=An example of an char separated table <<---
Foo.Bar
1.2

```

yields:

### 10.17 Verbatim

The verbatim region inserts text as seen on the screen. Arguments:

**wrap=N** The wrap margin

#### Example 10.19: Verbatim

```
#Verbatim wrap=10 <<---
Long long long long long long long line

```

yields:

```
Long long
long long
long long
long line
```

## 10.18 Write

You have to set the “w” or “W” `allow` flag(see 4) in order to enable this region.

Write or append (append! option) an expanded template to a file.

If file is “-”, the body will be printed on STDOUT.

**Example 10.20: Write something to a file**

```
#Write file=test.txt <<---
Foo {arg: myvar} bar.

```

## 11 Commands

`deplate` scans a text for line-elements and parses lines for text-particles. Commands are situated at the line-element-level, macros (see below) on the text-particle-level.

```
#COMMAND [OPTIONS]: ARGS
```

*NOTE:* It doesn't all work yet. It's a matter of trial and error to find out if `deplate` already supports a specific command. Most of this is untested and will fail or not work at all.

OPTIONS have the form:

- OPTION!
  - set option to true
- OPTION=VALUE
- the characters “!” and “=” have to be escaped with a backslash

Commands are applied only if the “fmt” and “if” options/conditions are met – see 14.6.

### 11.1 Getting or setting data about the document

**#AUTHOR: TEXT** define the document's author

- Synonym: #AU
- Each #AUTHOR command should take only one author
- Or separate authors like this:
  - AUTHOR1; AUTHOR2 ...
  - AUTHOR1 and AUTHOR2 ...
  - AUTHOR1 & AUTHOR2 ...
  - AUTHOR1/AUTHOR2 ...
- An author's name can have the form
  - FIRSTNAMES SURNAME
  - SURNAMES, FIRSTNAMES
- Alternatively, you can define the name as #AUTHOR `firstname=FIRSTNAME surname=SURNAME note=`
  - If you attach a note this way, you have to call #AUTHORNOTE nevertheless in order to update the “authornote” clip

**#AUTHORNOTE: TEXT** define an author note

- Synonym: #AN
- The note should follow immediately the #AUTHOR command it is referring to

**#CAPTION[!] [extended!] [above!|below!]: TEXT** add a caption to the previous element

- Synonym: #CAP
- refers to the previous element
- can be attached to: Table, Figure, Heading
- By default, TEXT will be parsed with certain macros/particles disabled. With a bang “!” or the “extended!” option, the full set of rules will be applied. There is chance though, this results in invalid output in certain cases, which could be relevant if, e.g., the output format is DocBook.

**#DATE: [TEXT|now|today|none]** Define the document’s creation date.

- Use none if you don’t want to include a time stamp in the document’s meta data.
- now is the default value.

**#DEFCOUNTER id=NAME** Define a new counter. Known arguments:

**parent** The parent list/and counter

- Has the format: NAME.LEVEL
- If no level is provided only the top numbering is used.
- If no parent is provided, the counter is “global”.

**#DEFLIST id=NAME** Define a new list and an associated counter. The lists can be printed using the #LIST command (possibly depending on the formatter). The standard lists, which should not be redefined, are:

**toc** Table of Contents

**lot** List of Tables

**lof** List of Figures

Custom lists may have the following attributes, which default to the list name:

**counter** The counter name

**entity** The name of the entities that are registered in the list

**prefix** The prefix used for labels

**parent** See #DEFCOUNTER.

**#GET: ID or GET id=NAME** Insert a clip at block/element level – see also 10.11.

**#KEYWORDS: WORDS** Set the document’s keywords. This really is the same as #VAR id=keywords: WORDS. Keywords should be separated by [;] (preferably “;”).

**#LANG: language** Change the document’s language. The language argument is the language identifier of the corresponding module, i.e., de for German, ru for Russian, zh-cn for Chinese and Chinese with automatic whitespace . . .

- Unless you load the language module from within the document or load more than one language module, there is no need to explicitly set the document's language.

**#OPT: KEY=VALUE** Attach some metainformation to the previous element (see 14.4).

- Synonyms: #PROP, #PP

**#PUSH: KEY=VALUE ...** Basically the same as #VAR add=, : KEY=VALUE.

**#REGISTER list=NAME: CAPTION** Register the previous element in list NAME, using an (optional) caption.

**#SET id=NAME: TEXT** define a clip – see also 10.11

**#TITLE: TEXT** define the document's title; the same as #VAR: title=TEXT

- Synonym: #TI

**#VAL: NAME** retrieve an argument (used in DefRegion (p. 56))

- The XVAL command is a variant of this and can be used to insert preformatted text.
- See 12.1 for additional arguments.

**#VAR: KEY=VALUE ...** define a document option (see 14.1)

- Synonym: #DOC (for document variable/option)
- You can also write: #VAR id=KEY: VALUE which is basically the same as the above form but you don't have to escape colons in the value string with a backslash.

## 11.2 Flow control, document management

**#BEGIN: ID and END: ID** when reading from stdin (and when using the command line option --loop), you can use these two pseudo commands to fragment the input; if the first line matches #BEGIN: ID, the input will end at the line #END: ID

- If you define the variable stdoutSeparator, its content will be added to each output document sent to stdout.

**#IF: TEST, ELSEIF: TEST, ELSE, ENDIF** include the following lines only if TEST is okay

- Operators: ==, !=, =~, !=~
- The fmt variable is matched against the current formatter's name
- A test is one of (VAR refers to a document option):

- fmt==FORMAT
- fmt!=FORMAT
- fmt=~FORMAT
- fmt!=~FORMAT
- VAR, VAR! (i.e., VAR is set and non-null)
- VAR==VALUE
- VAR!=VALUE
- VAR=~VALUE



– VAR!=~VALUE

- If “:” is set by the `--allow` command line option, you can access internal options by prepending the name with a colon (“:”). See 1 for possible option names.

**#INCLUDE: FILENAME** textually include a file

- Synonym: `#INC`
- `FILENAME` is a relative file name
- The file is searched for in:
  1. `./deplate.rc/FORMATTER/`
  2. `./deplate.rc/`
  3. `./`
  4. `~/.deplate/lib/FORMATTER/`
  5. `~/.deplate/lib/`
  6. `~/.deplate/`
  7. `DATADIR/deplate/lib/FORMATTER/`
  8. `DATADIR/deplate/lib/`
  9. `RUBYLIB/deplate/lib/FORMATTER/`
  10. `RUBYLIB/deplate/lib/`
- Arguments:
  - file=FILENAME** Alternatively you can use the `file` argument. If an explicit file argument and an anonymous argument are given, the explicit one overrides the anonymous one.
  - var=VARIABLE** You can also “include” the contents of a variable.
  - inputFormat=INPUTFILTER (experimental)** Set the input filter (see 6) for the included file.
  - \$embeddedTextRx=REGEXP** Transform the included text using this regular expression. This could be useful when including source files. `embeddedVerbatim` can be used to define what type of region should be used for formatting other text.
  - \$embeddedVerbatim=REGIONNAME** When using `embeddedTextRx`, use this region for formatting non-embedded text (e.g., `Code`)
  - \$codeSyntax=SYNTAX** Temporarily set the `codeSyntax` variable when using `embeddedVerbatim`
    - any other argument prepended with `$` will be used to temporarily set the variable of the same name.
- Variables:
  - You can also set variables in `includeVars` (a hash) that will be passed on to the command as if provided on the command line.

**#MODULE: NAME** require a module

- Synonym: `#MOD`

**#WITH file=FILE: REGION HEADER** create a region with input from file

## 11.3 Bibliography

**#BIB: FILENAME** a filename of a BibTeX database

**#MAKEBIB: [BIBSTYLE]** insert a formatted bibliography

- If the BIBSTYLE argument is missing, the `bibStyle` variable will be used. If the `bibStyle` variable doesn't exist, it will be set to the BIBSTYLE argument.
- In html mode, MAKEBIB works directly on the bibtex file. The main reason for this that it is easier to parse the bibtex file than to deal with the variety of bibtex-related styles and packages. Due to my simple-minded approach to parsing, MAKEBIB sometimes failes on some records and it doesn't cope well with entries other than books.
  - The html formatter stubbornly uses some kind of APA-based style.

## 11.4 Abbreviations, index

**#ABBREV [native!|ins!|plain!] DEFINITION: FULL TEXT** replace an abbreviation (defined as a word, a backtick symbol, *or* a regular expression) with the full title

- DEFINITION can be one of
  - `symbol=SYMBOL ...` a symbol will preceded by a backtick, i.e. if SYMBOL is `Sigma`, then the abbreviation matches `'Sigma`
  - `word=WORD ...` the WORD is surrounded by matches for word boundaries
  - `rx=REGEXP`
- if the option `plain!` is given, the full text won't be parsed
- if the options `native!` or `ins!` is given, the full text will be inserted as is

**#IDX: INDEX\_ENTRY|SPELLINGS...; OTHER\_ENTRY** add an index entry; use “|” to mark alternative spellings when using autoindexing; separate different entries with a semi-colon

- If autoindexing isn't switched off, you have to use this command only once to indicate that a certain word should be added to the index; all occurences downwards will be indexed automatically
- You can mark a word for autoindexing without inserting an index at the current position with the `#AUTOIDX` command

**AUTOIDX: INDEX\_ENTRY|SPELLINGS...; OTHER\_ENTRY** this can help avoiding certain problems with docbook output

- Use the following commands to selectively disable the automatically generation of an index

**NOIDX: INDEX\_ENTRY|SPELLINGS...; OTHER\_ENTRY** remove the *last* matching (auto)index entries

**DONTIDX: INDEX\_ENTRY|SPELLINGS...; OTHER\_ENTRY** avoid the *next* automatically generated index to be recorded

## 11.5 Dynamic text, elements

**#IMG [here!|top!|bottom!] [noGuess!] [bw=N|bh=N|w=N|h=N]: FILENAME** insert an image from a file

- Synonyms: **#IMAGE**, **#FIG**, **#FIGURE**
- Other options (depends on the formatter):
  - w=LENGTH**, **width=LENGTH** Image width
  - h=LENGTH**, **height=LENGTH** Image height
  - noGuess!** Usually deplate ignores any suffix and test the filename against a list of suffixes that supposedly work for the gived output format. Use this option to keep deplate from trying to be smart.
- for LaTeX output there are some extra arguments:
  - rotate=ANGLE** Rotate the image
  - bw**, **bh** define the bounding box in latex

**#LIST page! min=N max=N top=PREFIX sub!: [contents|toc|minitoc|tables|figures|index]** insert a table of [contents|tables|figures] or the index; the output depends on the formatter

- The **LIST: index** command should be placed in the last of the included files so that all indexes are defined.
- **toc** is a synonym for **contents**
- **minitoc** display the top headings only and uses the **shortcaption** argument if provided (in html output only)

**page!** Display the table/list on a new page

**min=N** List only elements with a level greater or equal N (HTML only)

**max=N** List only elements with a level less or equal N (HTML only)

**sub!** List only elements under the current heading (HTML only)

**top=PREFIX** List only elements the numbering of which begins with PREFIX (HTML only)

**levels=MIN..MAX, MIN.., ..MAX** Same as min and max

**#MAKETITLE [page!]** insert a title or titlepage

**page!** Create a title page (if supported by the output format). For LaTeX output, this adds “titlepage” to **classOptions**. If you use the **#VAR** command to set **classOptions** after the **#MAKETITLE** command, the variable will be reset. You could use **#PUSH** instead.

**#PAGE** start a new page

**#TABLE: FILE** read a table from FILE – takes the same options as the table region (see 10.16)

The commands **TITLE**, **AUTHOR**, **AUTHORNOTE**, and **DATE** define clips of the same name in lower cases (see the **get** macro for an example 12.1).

**Example 11.1: Commands**

```

#VAR: myvar=foo
#PUSH: myvar=bar
After: myvar={arg: myvar}

#IMG center! bw=110 bh=162: linked
#OPT fmt=latex: w=4cm
#CAP: A nice drawing

#SET id=foo: bar

Test SET: {get: foo}.

#IF: fmt==latex
This is in latex.
#ELSEIF: fmt=~^html
This is in html.
#ELSE
This is interesting.
#ENDIF

The file "calculations.txt" contains: 1 + 1. Let's see what R has to say
about this:
#WITH file=calculations.txt: R id=r_calculations: verb

#ABBREV sym=Sigma fmt=html ins!: Σ
#ABBREV sym=Sigma fmt=dbk if=noSgml ins!: Σ
#ABBREV sym=Sigma fmt=dbk if=sgml ins!: Σ
#ABBREV sym=Sigma fmt=latex ins!: Σ
The greek letter 'Sigma is called sigma.

#DEFLIST list=books title=Some Books parent=toc.1
#LIST: books
#DefElement id=books: ^BOOK:\s+(.*) <<--
{counter: books} {arg escapebackslash=2: 1}
\#OPT: style=book
\#REGISTER list=books name={arg escapebackslash=4 escape="!=": 1}
--
#Native fmt=html type=pre slot=css <<--
<style type="text/css">
<!--
div.book {
 margin-left: 40px;
 border-left: 5px solid #999999;
}
-->
</style>
--
BOOK: Max Frisch: Stiller

BOOK: William Shakespeare: Much Ado About Nothing

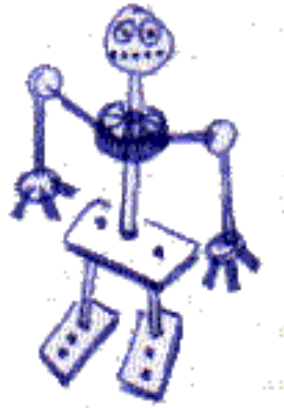
BOOK: Don \DeLillo: The Body Artist

yields:

 After: myvar=foo,bar
 Test SET: bar.
 This is in latex.

```

**Figure 4:** A nice drawing



The file “calculations.txt” contains:  $1 + 1$ . Let’s see what R has to say about this:

```
[1] 2
```

The greek letter  $\Sigma$  is called sigma.

## Some Books

**11.1** Max Frisch

**11.2** William Shakespeare

**11.3** Don DeLillo

11.1 Max Frisch: Stiller

11.2 William Shakespeare: Much Ado About Nothing

11.3 Don DeLillo: The Body Artist

## 12 Macros

Curly braces should be escaped with a backslash (i.e.,  $\backslash\{$  and  $\backslash\}$ ), as they usually mark macros:

```
{COMMAND [OPTIONS]: ARGS...}
```

*NOTE:* Macros cannot cross paragraph boundaries, i.e., they must not contain empty lines. Using newlines in a macro argument is useless, as the macro text will be collapsed to one single line.

*NOTE:* These aren’t macros in the sense of read-time expansions/replacements. They are called macros here because the (original) term `commands` is now used for `#CMD` type of commands only.

*NOTE:* If the macro name is unknown, a clip or a variable will be used instead. The order is: macros > clips > variables.

*NOTE:* Some macros parse their arguments, which then can contain other macros – e.g., `list` and `item`. “Parsing” is done with a simple nested regular expression. This appears to work quite well but could yield unintended results in some situations. The maximum of nested macros currently

is 5 (which should really enough due to the primitivity of the macro language). If you want more (e.g., 10), add this to your config file:

```
Deplate::Particle::Macro.build_rx(10)
```

### Example 12.1: Macro written in Ruby

```
#Ruby context=ruby <<--
class Deplate::Macro::Test < Deplate::Macro
 register_as 'test'
 def setup(text)
 @elt = @deplate.parse(@container, '<' + text + '>')
 end
end
end
--

Bla {test: [1
 {test: 2
 {test: 3
 {test: 4}
 }
 }
 inner bla
 {test: a
 {test: b
 {test: c
 {test: d}
 }
 }
 }
]}
}
Bla
```

yields:

```
Bla <[1 <2 <3 <4>>> inner bla <a <b <c <d>>>>>]> Bla
```

*NOTE:* You could run into problems if an macro argument contains unbalanced curly braces. It should work by preceding them with a backslash, but ...

## 12.1 Getting or setting data about the document

**Attributes:** {**attrib: KEY=VALUE ...**} Change the previous particle's (inline element) attributes; this command does on a text level what #OPT does for block elements

**Clips, variables:** {**get [default=TEXT]: ID**}, {**get id=ID [default=TEXT]**} see also 10.11; access "clips", e.g. the document's author: {get: author}

**Counters:** {**counter [depth=N] [delta=N]: NAME**} access a counter; see 11.5 for an example

**Element properties:** {**eprop ...**} Set the container element's properties (as if using the #PROP command).

**Document variables:** {**var: ID**} access document variable. See also the comments below on the **arg** macro. The difference between **var** and **arg** is that the **var** macro returns a value in a format can read.

**Element options:** {**opt:** **ID**} access element (paragraph, table etc.) options

**Retrieving variables :** {**arg:** **NAME**} Retrieve an argument (used in DefRegion (p. 56) or similar)

- Variables can also be accessed as in {**varname**} which will expand to {**arg: varname**}. {**varname!**} will expand to {**var: varname**}. Thus, {**varname capitalize!**} is equivalent to {**arg capitalize!: varname**}.
- The **xarg** macro is a variant of this that can be used to retrieve preformatted text.
- Synonyms: **val**, **xval**
- If the **. allow** flag is set, the **NAME** may have the form **NAME[METHOD]** to send methods to an object.
- Options:
  - join=SEPARATOR** Join the values of strings and hashes
  - values! keys!** Return a hashes values of strings
  - sub=/pattern/replacement/** Replace a pattern (the **'/'** is an arbitrary separator in this example; this can be any character)
  - tr=/pattern/replacement/** Character translation
  - uppercase!** Put the text in upper case letters
  - downcase** Put the text in lower case letters
  - capitalize!** Capitalize the text
  - escape="CHARACTERS"** Escape some characters with a backslash
  - escapebackslash!** Escape backslashes with a backslash
- The execution order of the text transformations arguments is fixed: **sub** > **tr** > **uppercase** > **downcase** > **capitalize**

**Localized messages:** {**msg:** **TEXT**} Insert a localized message (can be used in templates).

## 12.2 References, labels, index

**Anchor:** {**anchor:** **NAME**} Inserts an anchor at this very position; this should only used in inline position within a list item, a paragraph etc. It should be also used within lists or similar compound elements.

- Synonyms: **label**, **lab**

**Comments:** {**cmt:** **TEXT**} insert an empty string

**Index:** {**idx:** **NAME**} Inserts an index at this very position; this should only used in inline position within a list item, a paragraph etc. It should be also used within lists or similar compound elements.

**Reference:** {**ref** [**prefix=TEXT**] [**p!**]: **ANCHOR**} insert a reference to an anchor

**p** Reference to page

**prefix** Prepend this text to the reference (default: non-breaking space)

## 12.3 Bibliography

**Citation:** `{cite [p=PAGE] [n=NOTE] [np!] [y!]: ID1[;ID2...]}` output depends on the formatter

**np** No parentheses

**y** Year only

**p** The page number

**n** A note to be inserted before the citation (but within the parentheses)

## 12.4 Textstyles

**Code:** `{code: CODE}` or `{'CODE'}`, or `{verb: CODE}` the same as `'CODE'`

**Emphasized text:** `{_TEXT}` or `{em: TEXT}` or `{emph: TEXT}` emphasized text (the same as `__TEXT__`)

**Plain text:** `{plain: UNPARSED TEXT}` or `{\UNPARSED TEXT}` insert the text without interpreting deplate markup

**Subscript:** `{sub: TEXT}` or `{,TEXT}` Put TEXT in subscript

**Superscript:** `{super: TEXT}` or `{sup: TEXT}` or `{^TEXT}` Put TEXT in superscript

**Stacked:** `{stacked:{:SUPER}{:SUB}}` or `{%{:SUPER}{:SUB}}` Print SUPER on top of SUB (the HTML formatter uses css inline tables, which are displayed as expected with Firefox/Mozilla)

**Text:** `{text: TEXT}` or `{:TEXT}` Parsed TEXT. This macro can also be used to attach a style to some inline text.

**Mark 1st occurrence:** `{~text}` `{mark1st: text}` Mark TEXT, when the text is inserted for the first time. If the variable `mark1stStyle` is defined, this style will be used. Otherwise, the string's first occurrence will be printed in upper case letters. Arguments

**text=TEXT** Text used for lookup

**alt=STYLE** Mark other occurrences with this style

**always!, anyway!** Always mark as 1st occurrence, even if it isn't.

**Case:** `{downcase: TEXT}` `{upcase: TEXT}` `{capitalize: TEXT}` Change TEXT's case

## 12.5 Dynamic text, particles

**Date:** `{date: [format string|now|time|today|month|year]}` the format string uses ruby's `strftime` method.

**Footnote:** `{fn: ID}` or `{fn id=ID: TEXT}` The first form refers to a footnote defined by a Fn or Footnote region. The second form defines the footnote inline. The output depends on the formatter. The footnotes generated by the html formatter are compatible with OpenOffice.

**Images:** `{img [IMG arguments]: FILE}` insert an image; takes the same options as IMG (see 11.5)



**Inline LaTeX:** `{ltx id=ID [other Inlatex options]: LATEX CODE}` Insert small pieces of latex code.

- See also 10.8.

**Pagnumber:** `{pagnumber [hd!]: [TEXT]}` Returns the current page number. The result depends on the formatter. TEXT is appended to the pagnumber if the result is non-empty.

**HTML** Insert nothing

**LaTeX** Insert `\thepage` if the macro isn't used in a header region (the optional `hd!` argument)

**LaTeX + koma** Insert `\pagemark`

**Small lists:** `{list type=[ol|ul|dl]: ITEMS}` A convenience macro for inlining small lists into other elements. Items can be defined with the following two macros:

- `{item: TEXT}`
- `{term id=TERM: TEXT}`

**Native text:** `{ins: LITERALLY INSERTED TEXT}` e.g., `{ins fmt=html: &lt;&lt;}`

**Mathematical text:** `{math [block!]: LATEX FORMULA}` insert a small mathematical expression in latex syntax

- Synonyms: `$`
- If the `block!` argument is supplied, the formula is wrapped in `\[formula\]` instead of `$formula$`.
- See also 8.2.14 and 10.8.
- You can set the variable `mathPrelude` in order to prepend some LaTeX code to all mathematical formulae

**Newline:** `{nl}` start a new line

**Ruby code:** `{ruby [context=CONTEXT] [alt=ALTERNATE OUTPUT]: RUBY CODE}` • if the evaluation of ruby code is disabled, the text given in the alt option or an empty string will be inserted

- a sequence of ruby commands must be separated by semicolons
- for the CONTEXT argument please see 10.14

### Example 12.2: Macros

Footnotes:

```
Foo bar{fn: x} foo bar.
```

```
#Fn: x <<EOFN
```

```
 Ceci n'est pas une note.
```

```
EOFN
```

Reference:

```
Bar is bar.
```

```
#bar
```

```

Foo is foo (see also{ref: bar}).

Citation:
#BIB: deplate.bib
Mynona's{cite y! n=e.g.,: mynona:p1} real name was Salomo Friedlaender.

% You can find the bibliography down below. The command is commented out
% in order not to confuse nervous docbook tools.
% #MAKEBIB plain!: cbib-apa-1

Comments:
This line{cmt: here} contains a comment.

Date:
{date: today}

Ruby:
{ruby alt=ruby disabled: 1 + 1}

Image:
An inline {img rotate=90 h=40: linked} image (which is rotated in LaTeX
output).

LaTeX:
Test ({ltx: $\sum_{i=n}^n x_i$ }) the ltx macro.

Text formatting:
Foo{,bar & bar} is foo, but bar{^foo & foo} is bar.
Foo {_with_} or {plain: ''without''} bar.

{~Tom} says: "My name is {~Tom}."

{upcase: Foo bar}, {downcase: Foo bar}, {capitalize: Foo bar}.

x{:{2}{:a}} = 1

#IF: fmt!=latex
Small lists (this example doesn't work in latex):
| {list: {item: foo} {item: bar}} \
| {list type=dl: {term id=Foo: Foo is foo} {term id=Bar: Bar is bar}} \
|
| {list: {item: here} {item: there}} \
| {list type=dl: {term id=Here: Here is here} {term id=There: There is there}} \
|
#ENDIF

If you allow ".", you can could do something like this: This is the
paragraph with the id ''testParagraph1''.
#OPT: id=testParagraph1

The element with id=testParagraph1 is a {let t=object(id=testParagraph1):
{arg asString!: t.class}}.

#VAR: varname=Lala
#CLIP id=theSinger: __Maestro__ Maximo
{varname} is foo. {arg: varname} is foo.

{varname!} is bar. {var: varname} is bar.

```

`{theSinger} sings {varname!}`.

yields:

Footnotes: Foo bar<sup>1</sup> foo bar.

Reference: Bar is bar.


Foo is foo (see also 12.5).

Citation: Mynona's (e.g., 1980) real name was Salomo Friedlaender.

Comments: This line contains a comment.

Date: 10. Jul 2008

Ruby: 2

Image: An inline  image (which is rotated in LaTeX output).

LaTeX: Test ( $\sum_{i=n}^n x_i$ ) the `ltx` macro.

Text formatting: Foo<sub>bar&bar</sub> is foo, but bar<sup>foo&foo</sup> is bar. Foo `_with_` or `”without”` bar.

TOM says: “My name is Tom.”

FOO BAR, foo bar, Foo bar.

$x_a^2 = 1$

If you allow `.”`, you can could do something like this: This is the paragraph with the id `testParagraph1`.

The element with `id=testParagraph1` is a `Deplate::Element::Paragraph`.

Lala is foo. Lala is foo.

“Lala“ is bar. “Lala“ is bar.

*Maestro* Maximo sings “Lala“.

## 13 Skeletons

Skeletons look a lot like macros (e.g. `{{foo}}`) and they behave very much like included documents (using the `#INC` command), but they are expanded before actually parsing the input, i.e. they provide some kind of pre-processor macro expansion facility.

In order to use a skeleton you have to create a file

- `deplate.rc/lib/NAME`

or (this one will be preferred if it exists)

- `deplate.rc/lib/FORMATTER/NAME`

and enable the skeleton on the command line by using the `--skeleton` command line option.

Some skeletons (currently only one) are “special”, i.e. not file based but are processed by `Deplate::SkeletonExpander#skeleton_#{SPECIAL}(args)`, where `args` is a hash:

---

<sup>1</sup>Ceci n'est pas une note.

`{{id: NAME}}` This skeleton inserts hopefully unique ID strings that can be used in templates for defining regions.

Skeleton are expanded in place. I.e. the first line of the expansion text is inserted right where the skeleton marker appears in the text. If the expansion begins with an element (a command, a region etc.), the first character in the file should be a newline.

Skeletons are handled like templates (see 5.5.2). This means:

- All macros and commands available in templates can be used.
- In certain situations, commands and regions that are valid within templates thus have to be preceded with a backslash in certain conditions in order to avoid premature expansion.

### Example 13.1: Skeletons (1)

deplate.rc/skeletons/foo:

```
{arg: which} foo is {arg: @body}.
```

Input:

```
Bla bla. {{foo which=This: bar}} Bla bla.
```

yields:

```
Bla bla. This foo is bar. Bla bla.
```

### Example 13.2: Skeletons (2)

deplate.rc/skeletons/bar.html:

```
\#Native <<{{id: bar.form}}
<form action="" method="get">
<p>Name:
<input name="name" type="text" size="30" maxlength="30" value="{arg: name}"/>
</p>
<p>
<button name="submit" value="1" type="submit">Send</button>
</p>
</form>
{{id: bar.form}}
```

deplate.rc/skeletons/bar:

```
Name :: {arg: name}
```

Input:

```
Bla bla.
{{bar name=Io mio}}
Bla bla.
```

yields:

```
Bla bla.
```

```
Name Io mio
```

```
Bla bla.
```

## 14 Variables and options

### 14.1 Document variables

Basically, there are two ways to set a variable:

1. The `#VAR` command
2. In the `deplate.ini` file

User variables that have no special meaning should begin with an underscore. In restricted input mode, setting variables not beginning with an underscore is disabled.

Variables with all upper-case letter names should be considered as read-only variables. These are set by `deplate` and are not meant to be modified by the user.

Please see 14.6 to find out how to set variables for specific output formats only or depending on another variable's value.

#### Example 14.1: Defining variables

Define a `baseUrl` but only when generating multi-file html output:

```
#VAR fmt=~html(site|slides): baseUrl=http://deplate.sourceforge.net/ baseUrlStripDir=1
```

Define some stuff that should go to the html css section:

```
#Var fmt=~html: cssExtra <<--
<style type="text/css">
 <!--
 @media print {
 #tabBar {
 visibility: hidden;
 max-height:0;
 }
 }
 -->
</style>
--
```

#### 14.1.1 Pre-defined variables

**env** The environment variables (HASH)

The following list mentions most variables that can be used to fine-tune some aspect of `deplate`'s behaviour. Please see 4, 5.3 or 11.1 to see how to set these variables.

#### 14.1.2 General

**authorSep** A string used to concatenate author names in some occasions (STRING, default: ;)

**autoindex** Enable automatically adding entries to index (BOOLEAN, default: false)

**autoBaseName** In multi-file output, deduce the output filename from the current input file (BOOLEAN, default: false)

**autoFileNames** Enable automatic generation of file names based on the top heading for multi-file output (BOOLEAN, default: false)

**auxiliaryDir** If defined, auxiliary files will be put into this subdirectory (STRING)

**auxiliaryDirSuffix** If defined, auxiliary files will be put into a subdirectory named after the base name of the root file plus this suffix (STRING)

**bibClickableCitation** Whether citations (eg in HTML output) are clickable hyperlinks (BOOLEAN, default=false)

**bibStyle** Default bibliography style (STRING)

**bibSepAuthors** The separator for author names (STRING, default="; ")

**bibSepLastAuthor** The separator for the last author name (STRING, default="; ")

**bibSepTwoAuthors** The separator between two author names (STRING, default="; ")

**bibSepName** The separator between prename and surname (STRING, default=", ")

**class** The document class; used in LaTeX (= document class) or HTML output (= CSS) (STRING)

**codeLineNumbers** Display line numbers (BOOLEAN, default: false)

**codeStyle** The default style used for formatting code regions; depends on the backend (STRING)

**codeSyntax** The default syntax for #Code regions (STRING)

**commentsShow** Display comments in the output (BOOLEAN = any comment or STRING = comments with this prefix/marker)

**description** A short description of the document (STRING)

**docBasename** The base name of the output document (STRING)

**efilter** Include only elements in the output that are tagged with one of these tags. **any** matched untagged elements. (COMMA-SEPARATED LIST)

**elementStyle** If set to 'block', the markup of styles changes. For LaTeX output, this make `deplate` use regions. You can set this for single elements too. (STRING)

**embeddedTextRx** A regular expression matching embedded text (submatch \$1) (REGULAR EXPRESSION)

**embeddedVerbatim** The region name that will be used for text not matching `embeddedTextRx` (STRING, default: Verbatim)

**encoding** The default encoding; dependent on the output format (STRING)

**HTML** ISO-8859-1

**LaTeX** latin1

**XML** UTF-8

- In order to set the encoding do: `#VAR: encoding=utf-8`

- To set the encoding for HTML output only: `#VAR fmt=html: encoding=utf-8`

**figureNumbering** How figures should be numbered (STRING)

**flat, document** One counter for all figures

**none** No numbering

**section (DEFAULT)** Per section

**floatAlign** The default alignment of floats; dependent on the output format (STRING)

**headings** The headings style (STRING)

**plain** Don't number headings

**homeIndex** The heading index that should be the home page in multi-file output (INTEGER; default: 0)

**hyperHeading** If set and a heading has an `url` option attached, turn the heading into a hyperlink; if set to "full", then heading is used as link text, which may result in invalid output; otherwise a button is attached to the heading (BOOLEAN or STRING)

**hyperHeadingButton** The text for hyperlinked headings (STRING; default:  $\Rightarrow$ )

**imgSfx** The default suffix for images (STRING, default: png)

**includeVars** The variables to be used for `#INC` commands (HASH)

**indexwiki** If "no", don't autoindex extended wiki links (STRING)

- `<+TBD+>`This will be subject of change

**inlatexPrelude** A string or array of strings that will be prepended to each inline LaTeX snippet

**inlatexHeight, inlineLatexHeight** Define the `h` parameter of images generated by an inline LaTeX snippet (NUMBER)

**itemizeMarkers** The markers for itemize lists (currently only used for plain text output) (COMMA-SEPARATED LIST)

**ltxPointsize** The pointsize used for LaTeX snippets; this can be overridden by a per-element 'point-size' argument; see also `ps2imgRes` (p. 88) (NUMBER, default: 10)

**ltxSfx** The suffix of images generated by an inline LaTeX snippet (STRING, default: dependent on the output format)

**keywords** The keyword to categorize the document; either an array of strings or a list separated by semi-colons

**levelshift** Shift heading levels (NUMBER)

**ltxPrelude** Like `inlatexPrelude` but used for the `ltx` macro only

**mandatoryID** If set, elements that create auxiliary files must have an ID (BOOLEAN)

**mark1stStyle** If set, use the style to markup the `mark1st` macro (STRING)

**mathPrelude** Like `inlatexPrelude` but used for the `math` macro only

**noExplicitNumbering** Ignore the explicit numbering in lists (BOOLEAN)

**particleStyle** If set to ‘span’, the markup of styles changes. For LaTeX output, this make `deplate` use proper commands. You can set this for single particles too. (STRING)

**pdfOutput** Automatically set when using pdf output (the `--pdf` command line switch) (BOOLEAN)

**pfilter** Include only particles in the output that are tagged with one of these tags. `any` matched untagged particles. (COMMA-SEPARATED LIST)

**prefixID** Prefix autogenerated names for auxiliary files with this string; if not defined, use the current document’s base name (STRING)

**ps2imgRes** The resolution used for converting postscript files to bitmaps, e.g., by `ps2ppm` (NUMBER, default: 120)

**refButton** String to be used to represent the wiki references (STRING)

**rPrelude** Like `inlatexPrelude` but used for `Img` regions of type `R`

**rScanTable** If non-nil, guess column breaks in `R` tables (BOOLEAN)

**stdoutSeparator** A string printed after each output unit when using multi-file output and when redirecting the output to `STDOUT` (STRING)

**styledTags** If set, tags will automatically style an element with style “`TAG#{tagname}`” (BOOLEAN)

**suffix** The suffix to be used when guessing target names referenced by wiki names (STRING)

- `<+TBD+>`This will be subject of change

**tableNumbering** How tables should be numbered (STRING)

- flat, document** One counter for all figures
- none** No numbering
- section (DEFAULT)** Per section

**tableStyle** The default style for tables (STRING)

**tabwidth** The tab width used when expanding whitespace (NUMBER, default: 4)

**tag** Tag all elements with these tags (COMMA-SEPARATED LIST)

**template** The default template for the output (STRING)

**template\_version** Which template engine to use (NUMBER, default: 2)

**useParentSuffix** When referring to other files, use the source file’s suffix as fallback strategy (BOOL, default; false)

**verbatimMargin** Wrap verbatim regions at this width (NUMBER)

**wrapMargin** Wrap margin; set to 0 to prevent text wrap (NUMBER, default: 72)



### 14.1.3 Output Format

#### HTML

**baseUrl** The document's base url (STRING)

**baseUrlStripDir** Number of directories to remove when adding a file name to **baseUrl** (NUMBER)

**bodyOptions** The arguments passed to the body tag (STRING)

**css** The document's CSS file(s) (COMMA-SEPARATED LIST)

- The output media can be defined in the form: CSS|MEDIA

**cssExtra** Some extra text to be inserted at css position (STRING or ARRAY OF STRINGS)

**cssInclude** Include the contents of the CSS file in the HTML output (BOOLEAN)

**docNavbar** If true, display a navigation bar (BOOLEAN)

**explorerHack** Code to deal with browser-specific stuff; inserted at head position (STRING or ARRAY OF STRINGS)

**headExtra** Other HTML code to be inserted at head position (STRING or ARRAY OF STRINGS)

**homeButton** The "home" button in the navigation bar (STRING, default: [-])

**htmlAuxUrl, htmlCssUrl, htmlImgUrl** Format string that defines the URLs for images, CSS etc. (STRING)

**metaDataExtra** Other HTML code to be inserted at head position (STRING or ARRAY OF STRINGS)

**newsFeed** An array of strings defining news feeds for the current site. (STRING or ARRAY OF STRINGS)

- Entry format (please enclose the arguments in quotations marks):

- `rss='URL' title='Title'`
- `atom='URL' title='Title'`
- `href='URL' title='Title'` (defaults to rss)
- `type=rss href='URL' title='Title'`
- `type=atom href='URL' title='Title'`

**nextButton** The "next" button in the navigation bar (STRING, default: >>)

**nextKey** The keycode for jumping to the next page (NUMBER, default: 16)

**noBindKeys** If true, don't bind keys for jumping to the previous or next page (BOOLEAN, default: false)

**noGenerator** If true, don't add a generator tag (BOOLEAN, default: false)

**noNavMenu** If true, don't add a menu to the navigation bar (BOOLEAN, default: false)

**noSwallow** If true, don't swallow paragraphs in html-slides output format (BOOLEAN, default: false)

**noTabBarButtons** If true, don't add buttons to the navigation bar (BOOLEAN, default: false)

**pageIcon** The page icon tag (STRING)

**prevButton** The "previous" button in the navigation bar (STRING, default: <<)

**prevKey** The keycode for jumping to the previous page (NUMBER, default: 8)

**shortcutIcon** The shortcut icon tag (STRING)

**stepwiseDisplay** Whether to display a page step by step (BOOLEAN, default: false)

**stepwiseBegin** Number of initially visible elements (INTEGER, default: 1)

**stepwiseContinous** Automatically move to the next page if all elements are displayed. If the value is `confirm` the user will be queried before moving to the next page. (MIXED, default: false)

**stepwiseKey** Key code for revealing the next element (COMMA-SEPARATED LIST; default: 34 = 'PageDn')

**styleExtra** Some extra CSS information that will be wrapped in a style tag (STRING or ARRAY OF STRINGS)

**subToC** Print a table of contents for the current section after top level headings (BOOLEAN, default: false)

**tabBar** Items in the navigation bar (ARRAY OF STRINGS, default: [{"auto"}])

**tabBarButtons** If true, don't add buttons to the navigation bar (BOOLEAN, default: true)

- <+TBD+>Conflict with `noTabBarButtons`?

**tabEqualWidths** If true, tabs will have equal width (BOOLEAN, default: false)

**tabBarHomeName** The display name of the front page in the navigation bar (STRING, default: <+TBD+>)

**tabBarSep** The separator between display names and URLs in manual definitions of the navigation bar (STRING, default: |)

**tableWidth** The default table width

**useDublinCore** If true, use Dublin core tags instead of the standard meta tags (BOOLEAN, default: false)

**LaTeX**

**bookClass** Classes that use a `\chapter{}` command (COMMA-SEPARATED LIST or BOOLEAN)

**classOptions** Options passed on to the document class (COMMA-SEPARATED LIST)

**typeareaDIV, DIV (koma)** If set, use KOMA script's typearea package; use this variable to define the proportion of the printed area on a page; see the KOMA script manual for details (NUMBER)

**typeareaDIV\_**, **DIV\_ (koma)** Same as the above but the parameter is set as a document class option.

**floatHere** If true, add a `h` flag to floats (BOOLEAN, default: false)

**linespread** If set, define the `linespread` parameter (NUMBER)

**noteSize** The font size for margin notes (NUMBER, default: footnotesize)

**texLists** If set to `wide`, the generated source will contain more linebreaks. Otherwise the lines will be more condensed.

**useBooktabs** If true, use the `booktabs` package (BOOLEAN, default: false)

## Docbook

**copyrightYear** The year field in the copyright section (STRING)

**dbkEntities** Entities to be added to the `doctype` tag (ARRAY OF STRINGS)

**dbkVersion** The docbook version to use (STRING, default: 4.2)

**manvol** The manual volume to use for `dbk-ref` output (NUMBER, default: 1)

**refentry** The reference category used by for `dbk-ref` output (STRING)

**sgml** If true, try to be SGML compliant; useful when using tools like `openjade` (BOOLEAN, default: false)

**tableFrame** The frame parameter for tables (STRING, default: topbot)

## Plain Text

**asciiArt** Use `jave` to convert images to ascii representations; possible values: currently only `jave` (STRING, default: `jave`)

**imgAlt** Fallback display name for images (STRING)

### 14.1.4 Module

#### Anyword

**anyword\_catalog** A catalog file containing automatically linked words (STRING)

**anyword\_list** A comma-separated list of automatically linked words (STRING)

**anyword\_pattern** A glob pattern for file names in the current directory the base names of which will be added to the list of automatically linked words (STRING)

**anyword\_suffix** The suffix to use when scanning the directory using `anyword_pattern` (STRING)

## CJK

**cjk\_encoding** The default encoding (STRING, default: GB)

**cjk\_family** The default font family (STRING, default: gbsn)

**noSmartBlanks** If true, guess whether a whitespace should go into the output (BOOLEAN, default: false)

### entities

**entities** List of entities names (COMMA-SEPARATED LIST, default: general)

### html-headings-navbar

**headingsNavbarMaxLevel** Add heading up to this level to the navigation bar (NUMBER)

**html-sidebar** Deprecated module.

**mouseThresholdIn** (NUMBER)

**mouseThresholdOut** (NUMBER)

**navGif** (STRING)

### latex-verbatim-small

**verbSize** Font size for verbatim regions (NUMBER, default: footnotesize)

### Mark External URLs

**mailtoIcon** The file name of the e-mail icon (STRING, default: mailto.png)

**urlIcon** The file name of the external URL icon (STRING, default: url.png)

**markerInFrontOfURL** Put the icon in front of the URL (BOOLEAN, default: false)

### navbar-png

**buttonsColour** The subtype of the button images

**buttonsFileFormat** The image suffix (STRING, default: png)

**buttonsHighlight** Use a rollover effect (BOOLEAN, default: false)

### Obfuscate E-Mail

**noObfuscatedNoscript** If true, put only the name into the noscript tag (BOOLEAN, default: false)

**obfuscatedNoscriptAt** How to display the @ sign in the noscript region (STRING, default: AT )

**obfuscatedNoscriptDot** How to display periods in the noscript region (STRING, default: DOT )

## **pstoedit**

**pstoeditArgs** Additional command line arguments (STRING)

## **Smileys**

**smileySfx** The suffix for smiley image files (STRING, default: png)

## **validate-html**

**noCssValid** If true, don't add a button for CSS validation (BOOLEAN, default: false)

**noHtmlValid** If true, don't add a button for HTML validation (BOOLEAN, default: false)

## **XMLRPC**

**xmlrpcAllow** A comma-separated list of valid IP addresses or regular expressions matching valid addresses

**xmlrpcPath** The path for accessing this xmlrpc server (STRING, default: /deplate)

**xmlrpcPort** The xmlrpc server port (NUMBER, default: 2000)

**xmlrpcReuseInterpreter** If non-nil, reuse the converter as if the requests were part of one big document (BOOLEAN, default: nil)

### **14.1.5 Legacy**

**legacyDefine1** Make #Define type of regions behave as in version up to 0.8 (alternatively, you can pass a noTemplate! argument to the region)

legacyFor1 :: Make #For behave as in version up to 0.8 (alternatively, you can pass a noTemplate! argument to the region)

## **14.2 Strings, booleans, arrays, and hashes**

A variable can be either a string, a boolean, an array, or a hash. The type is defined by the way how it is defined.

Using booleans actually makes only sense in conjunction with **if** clauses as, in other situations, the value will be represented as the strings "true"/"false".

String:

```
#VAR: var=value
```

Boolean:

```
#VAR: var!
#VAR: noVar!
#VAR: var=true
#VAR: var=false
```

Array:

```
#Var: id=var <<---
Value 1
Value 2

```

```
#VAR: var[]=value1
#VAR: var[]=value2
```

Hash:

```
#VAR: var[field1]=value1
#VAR: var[field2]=value2
```

The values can be retrieved using the `var` or `val` macro.

### Example 14.2: Variable types

```
#VAR: stringVar=Foo
```

```
#VAR: boolVar1!
#VAR: noBoolVar2!
#VAR: boolVar3=true
#VAR: boolVar4=false
#VAR: isNoBoolVar1=\true
#VAR: isNoBoolVar2=\false
```

```
#Var id=arrayVar1 <<---
Foo
Bar

```

```
#VAR: arrayVar2[]=foo
#VAR: arrayVar2[]=bar
```

```
#VAR: hashVar[field1]=Foo
#VAR: hashVar[field2]=Bar
```

```
| stringVar = | {val: stringVar} | {stringVar!} |
| boolVar1 = | {val: boolVar1} | {boolVar1!} |
| boolVar2 = | {val: boolVar2} | {boolVar2!} |
| boolVar3 = | {val: boolVar3} | {boolVar3!} |
| boolVar4 = | {val: boolVar4} | {boolVar4!} |
| isNoBoolVar1 = | {val: isNoBoolVar1} | {isNoBoolVar1!} (this is a string) |
| isNoBoolVar2 = | {val: isNoBoolVar2} | {isNoBoolVar2!} (this is a string) |
| arrayVar1 = | {val: arrayVar1} | {arrayVar1!} |
| arrayVar2 = | {val join=", ": arrayVar2} | {arrayVar2!} |
| hashVar = | {val join=", " values!: hashVar} | {hashVar!} |
| arrayVar1[0] = | {val: arrayVar1[0]} | {arrayVar1[0]!} |
| arrayVar2[1] = | {val: arrayVar2[1]} | {arrayVar2[1]!} |
| hashVar[field1] = | {val: hashVar[field1]} | {hashVar[field1]!} |
```

yields:

stringVar =	Foo	“Foo“
boolVar1 =	true	true
boolVar2 =	false	false
boolVar3 =	true	true
boolVar4 =	false	false
isNoBoolVar1 =	true	“true“ (this is a string)
isNoBoolVar2 =	false	“false“ (this is a string)
arrayVar1 =	[“Foo”, “Bar”]	[“Foo“, “Bar“]
arrayVar2 =	foo, bar	[“foo“, “bar“]
hashVar =	Foo, Bar	{“field1“=>“Foo“, “field2“=>“Bar“}
arrayVar1[0] =	Foo	“Foo“
arrayVar2[1] =	bar	“bar“
hashVar[field1] =	Foo	“Foo“

### 14.3 Template “services”

Some option names have a special meaning in that the content is dynamically generated. Most of them are specific to some output format.

“Services” are meant to generate small, atomic data. Calling a “service” can be distinguished from accessing a variable by adding braces at the end of the service name – which makes it look like a function call in many programming language, which could lead to the question, why I didn’t call it template functions but ... heck.

“Services” are provided by the formatter. I.e. they are some kind of formatter specific commands that can be used in templates.

You can use camel case names and underscores according to your liking. `someService` and `some_service` do the same thing.

Syntax:

```
serviceName()
serviceName(KEY1=VALUE KEY2! ...)
```

Standard services:

- `html`
- **`navigation_bar`** Returns a simple navigation bar with a menu and forward/backward buttons (see 8.5.3).
- `html-slides`
- **`tabBarLeft`, `tabBarRight`, `tabBarTop`, `tabBarBottom`** returns a tab bar with the table of contents
- **`progressPercent`** insert the progress as a numeric value
- **`progressBar`, `progressBarHorizontal`, `progressBarVertical`** returns a simple progress bar

These services are most useful for designing templates. The following example contains the template for the HTML online documentation.

#### Example 14.3: Templates

```
#PREMATTER
```

```

<div id="tabBar" class="tabBar">
 #ARG: tabBarLeft(progressBar! depth=2 depthInactive=1)
</div>
<div class="tabBodyFrame">
 <div class="tabBody">
 #BODY: -navbar_top -navbar_bottom
 </div>
 #POSTMATTER: html_pageicons_beg..html_pageicons_end
</div>

#POSTMATTER

```

## 14.4 Element properties

Using the #PROP command, you can attach some metainformation to the previous element.

```

* Heading
#PROP: id=myHeading

```

- when collapsing elements (e.g. compiling list items to one list), later options overwrite previous ones with the same name

For commands and regions, these options are part of the definition:

```

#COMMAND optionA=ValueA optionB=ValueB: foo

#Region optionA=ValueA optionB=ValueB <<--
--

```

So why is there an #PROP command, anyway? It's there to add more flexibility to elements that are defined in wiki-like markup.

For macros and particles (inline elements), the situation similar. The following two text lines are equivalent:

### 14.4.1 Global properties

Global properties can be defined in a global hash variable.

#### Example 14.4: Global properties

The following will tag the heading “Foo” as “heading” and the text in foo.txt with “secondLevel”.

```

#VAR: $ElementHeading[tag]=heading
#VAR: $INC[tag]=secondLevel

* Foo
#INC: foo.txt

```

## 14.5 Tags and filters

The global variable `efilter` defines which elements to include in the output. Untagged elements use an implicit `any` tag.

In the following example, we include text stored in a variable in order not to mess up (by filtering elements) the formatting of the main document.

#### Example 14.5: Tags and filters



```

#Var id=tagsExampleInput <<---
* Foo
#PROP: tag=heading

Bar says barly "Bar".
#PROP: tag=bar

Boo says booly "Boo".
#PROP: tag=boo

Foo says fooly "Foo".
#PROP: tag=foo

Something else.

#INC $levelshift=2 $efilter=heading,foo,any var=tagsExampleInput

```

yields:

### 14.5.1 Foo

Foo says fooly “Foo”.  
Something else.

## 14.6 Special arguments

The following arguments usually have special meanings. You shouldn’t use these names for macro arguments and the like.

`id=NAME` The element’s ID

`style=STYLE1,STYLE2,...` The element’s style

`swallow!` Don’t print

`add!` or `add=SEPARATOR` the following options will be appended to already existing options of this name using `SEPARATOR` or a blank (`add!`)

- the following example sets the element option “test” to “1,2”
- Example (which is equivalent to the `#PUSH` command):
  - `#VAR add=, : test=1`
  - `#VAR add=, : test=2`

`type=[body|preMatter|postMatter]` The document section where the element will be put

`slot=N` The slot where the element will be put

- If `N` is negative, the text will be put at the slot’s first position
- See also 15.1.

`indentation=[auto|none]` If an element is embedded in a list, you can change the way it is indented in the output.

**auto** Let the formatter decide

**none** Print left aligned

The **fmt** and the **if** options are special in that they are interpreted by **deplate** and prevent an element from being printed if they are not true:

**fmt=FORMATTER** Print the element/particle only if the current output formatter's name is **FORMATTER**

- Variants: `fmt=~FORMATTER,` `nofmt=UNMATCHED FORMATTER,`  
`nofmt=~UNMATCHED FORMATTER`
- Example:
  - `#VAR fmt=html: class=mycss`
  - `#VAR fmt=latex: class=report`

**if=VAR** Print the element/particle only if the test evaluates to true

- Variants: `if=noVAR,` `if=(VAR==VALUE),` `if=(VAR!=VALUE),` `if=(VAR=~VALUE),`  
`if=(VAR!=~VALUE)`
- the **if** clause should be put in parentheses or double quotes in order to avoid parse errors
- the element/particle will be inserted only if the document variable **VAR** is set or matches/doesn't match **VALUE**
- if the dooption begins with “no” and the next letter is in upper case, the dooption's name will be converted to lower case and tested against its non-presence; see 11.5 for an example
- Example:
  - `#TITLE if=(outcome==good): Hurray!`

## 14.7 Clips

Clips are formatted text that were saved for later use. In opposition to variables, which are formatted after insertion, clips are formatted independently from where they are used.

Some command generate clips automatically. These auto-created clips are:

**author** The document's author

**authornote** A note about the author

**title** The document's title

**date** The document's creation date

## 15 Internals

### 15.1 Document structure

The formatted **deplate** document is made up of a pre-matter, a body, and a post-matter each of which is an array of arrays of strings.

Important:  
It's possible  
that “clip”  
will disappear  
in a future  
release.

- Pre-matter
  - Slots
    - \* Array of strings
- Body
  - Slots
    - \* Array of strings
- Post-matter
  - Slots
    - \* Array of strings

Using the `docType` option, you can determine into which section an element goes. Using the `docSlot` option, you can define the element's position within a section.

In multi-file output, pre-matter and post-matter are the same for all files. The actual body usually begins, but this depends on the formatter, at position 90 of the pre-matter and ends at position 20 of the post-matter section. I.e. if you want to print some information in every file's body, put it somewhere in the pre-matter after position 90 or in the post-matter before slot 20.

The default slot for normal text is 50 in the body section. If slot is greater than 50, the element will be moved to the end of the document. If it is smaller, it will be moved to the beginning. The utility of this feature for the user is limited. It can come handy, if you want to put LaTeX-code into the preamble or if you want to customize the output in some more advanced way.

The slots can also be addressed via names:

- Pre
  - doc\_def** The document type definition
  - doc\_beg** The beginning of the document
  - head\_beg** The beginning of the document head
  - javascript** Javascript
  - mod\_packages** Packages loaded by a module
  - user\_packages** User-required packages
  - mod\_head** Additional header statements added by a module
  - user\_head** Additional header statements added by the user
  - head** Document header statements
  - user\_head** Document header statements added by the user
  - css** Style definitions
  - head\_end** The end of the document head
  - body\_beg** The beginning of the body definition
- Body
  - body** The body (standard slot)
  - footnotes** Footnotes

- Post

**body\_end** The end of the body definition

**doc\_end** The end of the document

The names can come handy when selectively filling in content into a template.

Any element can be located at any position in the document. This can be done by adding the option “slot” to an element. The value of “slot” can be a name, a number, or a mix of both like in “css+1” or “body-1”.

In general, you shouldn’t address slots by their number. It’s quite likely that this option will soon disappear.

Important:  
The use  
numbers  
deprecated  
and will  
disabled soon  
time in the  
future.

## 16 Extending deplate

A `deplate` document consists of elements (i.e., single lines or group of lines) that are made up of particles (sub-line-level text bits).

### 16.1 Line/Elements

Elements are processed in several passes:

- accumulate a new element; it’s type is inferred from the first line
- add other lines
- “finish” the element (unify with previous elements if applicable)
- process the element (at this point of time, all data/references etc. are known)
- output the formatted element (at this point of time, all elements were processed)

### 16.2 Text/Particles

Particles are processed in two passes:

- create an object and parse the text
- process the element and return a formatted string

#### 16.2.1 Symbols

If you don’t want to define symbols using `#ABBREV` in your source file (maybe because of performance considerations), you can make them available by creating a file like `~/.deplate/after/fmt/{FORMATTER}/mysymbols.rb`, which contains something like the following HTML example:

```
Set the context for where we want to store our symbol definitions,
here: HTML output format
class Deplate::Formatter::HTML
 # Create a hook function (the name must begin with
 # "formatter_initialize_")
 def formatter_initialize_my_symbols
 # Define our symbols. We could either create a hash and merge it
```

```

 # with the already defined @special_symbols or add each entry with
 # a single command as in this example
 @special_symbols['€'] = '€'
 @special_symbols['Ä'] = 'Ä'
 # We need to rebuild the regular expression used for tokenizing
 build_plain_text_rx
 end
end

Set the context: LaTeX output format
class Deplate::Formatter::LaTeX
 def formatter_initialize_my_symbols
 @special_symbols['€'] = '\EURtm{ }'
 @special_symbols['Ä'] = '\ "A{ }'
 build_plain_text_rx
 end
end
end

```

Be aware though that this file will not be loaded when using a HTML variant like the `htmlsite` formatter. If you want to make these symbols be available in all descendants of the HTML formatter, you will have to move this ruby code to `~/deplate/config.rb`.

### 16.2.2 Macros

For adding a new macro, you would have to put something like this into your `config.rb`<sup>2</sup>:

Or use the `simple_macro` function/method as in:

```
Deplate::Core.simple_macro("mymacro", %{text * 3})
```

The macro is now accessible.

```
Foo {mymacro: bar}
```

This would then produce:

```
Foo barbarbar
```

## 16.3 Formatters

Sometimes, the best way to change `deplate`'s output is to create a new formatter. Let's take the `html-snippet` formatter as an example. This formatter derives from the HTML formatter but has a different name and doesn't format paragraphs (the assumption is that this formatter is only used for small text snippets which are later on reused by an other application):

```
Make the HTML formatter known
```

```
require 'deplate/fmt/html.rb'
```

```
Inherit from the HTML formatter
```

```
class Deplate::Formatter::HTML_Snippet < Deplate::Formatter::HTML
```

```
 # Give this formatter a name (this usually is the file's base name
```

```
 NAME = "html-snippet"
```

```
 # Define an initialization hook that is called after creating a new
```

```
 # instance of this formatter.
```

---

<sup>2</sup>You could also use a ruby region 10.14 to define the command in place. This requires the `-x` command-line switch to be given, though.

```

Make sure we run in "included" mode as we don't want any HTML
preamble in the output when formatting single phrases that will
be reused by another application.
def formatter_initialize_snippet
 unless @deplate.options.included
 log(['Not run in included mode'], :error)
 log(['Set included mode'], :error)
 @deplate.options.included = true
 end
end

Override the default method for formatting paragraphs. In this
case, we simply return the element's content.
def format_paragraph(invoker)
 invoker.elc
end

Make the formatter known to "deplate".
class Deplate::Core
 declare_formatter Deplate::Formatter::HTML_Snippet
end

You could now save this code in, say, ~/.deplate/fmt/html-snippet.rb3 and invoke the newly
defined formatter from the command line:

> echo '__Foo__ "bar".' | deplate -f html-snippet --included -
Foo “bar”.

```

## References

- Mynona (1980): *Ich verlange ein Reiterstandbild. Prosa Band 1*. edition text + kritik.
- Venables, William N., Ripley, Brian D. (2003): *Modern Applied Statistics with R*. New York: Springer, 4 edn.

---

<sup>3</sup>This formatter is already included in the distribution. So, you don't have to actually save it in order to use it.

## Index

"template" input filter, 30  
"template" output filter, 27  
#ABBREV, 74  
#AN, 71  
#ARG, 57, 72  
#AU, 70  
#AUTHOR, 70  
#AUTHORNOTE, 71  
#AUTOIDX, 74  
#Abstract, 55  
#BEGIN, 72  
#BIB, 74  
#CAP, 71  
#CAPTION, 71  
#Code, 56  
#DATE, 71  
#DOC, 72  
#DONTIDX, 74  
#DefCommand, 56  
#DefCommandN, 56  
#DefElement, 56  
#DefMacro, 56  
#DefMacroN, 56  
#DefRegion, 56  
#DefRegionN, 56  
#Defc, 56  
#Defcn, 56  
#Defe, 56  
#DefineCommand, 56  
#DefineElement, 56  
#DefineMacro, 56  
#DefineRegion, 56  
#Defm, 56  
#Defmn, 56  
#Defr, 56  
#Defrn, 56  
#Doc, 60  
#ELSE, 72  
#ELSEIF, 72  
#END, 72  
#ENDIF, 72  
#Footer, 61  
#For, 60  
#GET, 71  
#Header, 61  
#Html, 65  
#IDX, 74  
#IF, 72  
#IMG, 75  
#INC, 73  
#INCLUDE, 73  
#Img, 61  
#Inlatex, 62  
#KEYWORDS, 71  
#LANG, 71  
#LIST, 75  
#Latex, 65  
#Ltx, 62  
#MAKEBIB, 74  
#MAKETITLE, 75  
#MOD, 73  
#MODULE, 73  
#Mingle, 21  
#NOIDX, 74  
#Native, 65  
#OPT, 72  
#PAGE, 75  
#PP, 72  
#PROP, 72  
#PUSH, 72  
#Qu, 65  
#Quote, 65  
#R, 66  
#Ruby, 68  
#SET, 72  
#Set, 66  
#Skip, 69  
#Swallow, 69  
#TABLE, 75  
#TI, 72  
#TITLE, 72  
#Table, 69  
#VAL, #XVAL, 72  
#VAR, 57, 72  
#Var, 60  
#WITH, 73  
#XARG, 72  
{}, 80  
{:}, 80  
{%}, 80

`{^}`, 80  
`{_}`, 80  
`{}`, 80  
`{~}`, 80  
`{anchor}`, 79  
`{arg}`, 57, 79  
`{attrib}`, 78  
`{capitalize}`, 80  
`{cite}`, 80  
`{cmt}`, 79  
`{code}`, 80  
`{date}`, 80  
`{doc}`, 78  
`{downcase}`, 80  
`{em}`, 80  
`{emph}`, 80  
`{eprop}`, 78  
`{fn}`, 80  
`{get}`, 78  
`{idx}`, 79  
`{img}`, 80  
`{ins}`, 81  
`{item}`, 81  
`{list}`, 81  
`{ltx}`, 81  
`{mark1st}`, 25, 80  
`{math}`, 81  
`{nl}`, 81  
`{opt}`, 79  
`{pagenumber}`, 81  
`{plain}`, 80  
`{ref}`, 79  
`{ruby}`, 81  
`{stacked}`, 80  
`{sub}`, 80  
`{sup}`, 80  
`{super}`, 80  
`{text}`, 80  
`{upcase}`, 80  
`{val}`, 79  
`{var}`, 57  
`{verb}`, 80  
`{xarg}`, 79  
`{xval}`, 79  
`{,}`, 80  
  
aft, 7  
allow, 13–15, 20, 32, 33, 43, 56, 58, 79, 83

André Simon’s highlight, 33  
ASCIIMathML, 40  
ASCIIMathML.js, 3, 9, 39  
autoFileNames, 15, 86  
auxiliaryDirSuffix, 15, 86  
  
Backslash, 42, 43, 45, 48, 51, 58, 70, 72, 77–79, 84  
Backtick, 42, 52, 74  
BibTeX, 27  
booktabs, 29, 91  
  
code-gvim71, 2, 8, 33  
codera, 2, 32  
commands, 41–43, 57, 58, 66, 72, 74, 75, 77, 81, 84, 87, 88, 95, 96  
config.rb, 14, 16–20, 40, 41, 52, 53, 101  
CSS, 16, 19, 25, 27, 80, 85, 86, 89, 90, 93, 99, 100  
Cygwin, 9  
  
D Benedetto & E Caglioti & V Loreto “Language Trees and Zipping“, 31  
DefRegion, 72, 79  
deplate, 1, 2, 6–8, 10, 13–16, 18, 20, 21, 23, 25, 30, 34, 37, 38, 41, 68, 73, 75, 80, 83–85, 93  
Deplate Homepage, 25  
deplate.ini, 14, 18, 19  
deplate.rc, 14, 16, 18, 73  
Deplate#user\_initialize, 31  
Dirk Holtwick’s “Guess language of text using ZIP“, 31  
DocBook, 2, 3, 7, 30, 34, 35, 40, 51, 74, 91  
document option, 31  
document option, 21, 40, 47, 51, 57, 72  
dokkit, 11  
dot, 18, 62  
dramatist, 2, 25, 29  
dvipng, 8  
dvips, 8, 9, 18, 62  
  
Elements, 3–5, 16, 23, 26, 28, 41–43, 51, 54, 59, 66, 68, 70–72, 75, 78, 79, 81, 83, 84, 86–88, 90, 96–100  
Emacs, 7, 55  
emacs-wiki, 7  
Example, 34–38, 43–46, 48, 51–60, 62, 64–70, 75, 78, 81, 84, 85, 94–98



Example in a box, 34  
 example-letter.pdf, 22  
 exerb, 9

Formatter, 17, 18, 21, 25–30, 35, 40, 48, 55,  
 59, 61, 64, 65, 71, 72, 74, 75, 80, 81,  
 95, 98, 99, 101, 102

gem package, 9  
 Ghostscript, 8, 9, 18  
 GNU General Public License, 9  
 Graphics, 29, 61, 62, 64, 65  
 graphicx, 29  
 GVIM, 9  
 gvim, 2, 33

highlight, 2, 9, 33, 53  
 Homepage, 25  
 hpricot, 8  
 HTML, 2, 3, 7, 13, 27, 28, 31–33, 35, 39–41,  
 48, 51, 54, 55, 59–61, 66, 68, 74, 75,  
 80, 81, 84–87, 89, 92, 93, 95, 100, 101  
 html-asciimath, 9  
 html-jsmath, 9  
 htmlslides, 3, 7, 13, 40  
 htmlslides-navbar-fh, 28  
 Hyperlink, 29, 45, 51, 87  
 hyperref, 29

ImageMagick, 8, 9, 29  
 index, 25

JavaScript, 28  
 Jave, 8, 30  
 jsMath.js, 3, 9, 40

kpsewhich, 8

LaTeX, 2, 3, 7–9, 13, 17, 18, 29, 39, 43, 48,  
 55, 62, 64, 69, 75, 76, 81, 92  
 latex-dramatist, 25  
 List, 3, 20, 21, 24–28, 32, 33, 38, 42, 44–46,  
 48, 60, 71, 72, 75, 79, 81, 85, 87, 88,  
 91–93, 96, 97

Macros, 4, 5, 16, 21–23, 29, 34, 36, 38, 39, 41–  
 44, 46, 56, 57, 61, 64, 66, 70, 71, 75,  
 77–81, 83, 84, 87, 88, 94, 96, 97, 101  
 Markup, 2, 3, 7, 8, 16, 20, 23, 25, 30, 36, 38,  
 41–43, 55, 80, 86–88, 96

math macro, 39  
 mathml, 3, 9, 40

natbib, 29  
 neato, 18, 62  
 Nukumi2, 41

Page number, 39, 61, 81  
 Particles, 2–5, 16, 36, 41, 59, 66, 70, 71, 78,  
 80, 88, 96, 98, 100  
 PDF, 20, 29, 35, 88  
 Peter Jipsen, 39  
 Php, 2, 28, 29, 35  
 PlainDoc (pd2tex), 7  
 play, 29  
 ps2imgRes, 87, 88

QBullets set, 35

R, 4, 9, 13, 18, 23, 55, 62, 66, 67, 77, 88  
 rdoc, 2, 7, 23, 24  
 regions, 41–43, 52, 56, 58, 59, 61, 84, 86, 88,  
 92, 93, 96  
 reStructuredText, 7  
 rote, 8  
 ruby, 7–10, 13–18, 20, 21, 23, 32, 37, 42, 52,  
 55, 56, 68, 80, 81, 101  
 Ruby interpreter, 8  
 rubygem, 9  
 rubygems, 9

Screen Play, 25  
 similar efforts, 8  
 sisu, 7  
 source, 23, 28, 32, 34, 37, 47, 54, 56, 59, 64,  
 66, 73, 88, 91, 100  
 Stage Play, 25

Term::ANSIColor, 9  
 This example, 34  
 tidy, 27  
 txt2tags, 7

URL, 51, 89, 90, 92  
 user\_initialize, 19  
 user\_setup, 19, 20

variable, 17–21, 26–30, 33, 34, 37, 45, 60, 61,  
 68, 72–75, 77, 78, 80, 81, 85, 90, 93,  
 95, 96, 98

viki, 7, 23, 41, 51  
Vim, 1, 7, 16, 23, 41, 51  
Vim viki plugin, 7, 8, 16

webgen, 8  
Whitespace, 3, 54, 71, 88, 92  
Wiki, 51, 52, 55, 87, 88, 96  
WikiName, 43

XHTML, 2, 28, 32, 33  
xhtml10t, 2, 9, 28  
xhtml11m, 40  
XML, 28, 30, 41, 86

yodl, 7