

# struktex.sty\*

Jobst Hoffmann

University of Applied Sciences Aachen, Abt. Jülich

Ginsterweg 1

52428 Jülich

Federal Republic of Germany

printed on March 7, 2018

## Abstract

This article describes the use and implementation of L<sup>A</sup>T<sub>E</sub>X-*package* `struktex.sty` for structured box charts (Nassi-Shneiderman diagrams).

## Contents

<b>1</b>	<b>License</b>	<b>2</b>	<b>5</b>	<b>Example file for including into the documentation</b>	<b>23</b>
<b>2</b>	<b>Preface</b>	<b>2</b>			
<b>3</b>	<b>Hints for maintenance and installation as well as driver file creating of this documentation</b>	<b>3</b>	<b>6</b>	<b>Some example files</b>	<b>23</b>
			6.1	example file no 1 . . . . .	23
			6.2	example file no 2 . . . . .	24
			6.3	example file no 3 . . . . .	25
			6.4	Example file no 4 . . . . .	29
<b>4</b>	<b>The User interface</b>	<b>5</b>	<b>7</b>	<b>Macros for generating the documentation of the <code>struktex.sty</code></b>	<b>31</b>
4.1	Specific characters and text representation . . . . .	6			
4.2	Macros for representing variables, keywords and other specific details of programming . . . . .	7	<b>8</b>	<b>Makefile</b>	<b>36</b>
4.3	The Macros for generating structured box charts	9	<b>9</b>	<b>Style File for easier input while working with (X)emacs and AUCT<sub>E</sub>X</b>	<b>41</b>

## 1 License

This package is copyright © 1995 – 2018 by:

---

\*This file has version number v2.3-17-ga8f1af9, last revised on 2018/03/07, documentation dated 2017/12/31.

Jobst Hoffmann, c/o University of Applied Sciences Aachen  
Aachen, Germany  
E-Mail: j.hoffmann\_(at)\_fh-aachen.de

This program can be redistributed and/or modified under the terms of the LaTeX Project Public License, distributed from the CTAN archives as file `macros/latex/base/lppl.txt`; either version 1 of the License, or (at your option) any later version.

## 2 Preface

St<sub>u</sub>kT<sub>E</sub>X has a long history. In development several different programs for version management were used—rcs, subversion and currently git—, all of which offer different possibilities to define version numbers. To correlate these different version numbers correctly in time, the following schema will be used: Version numbers of the form “v-⟨d⟩.⟨d⟩[⟨a⟩]” with ⟨d⟩ as a decimal digit and ⟨a⟩ as a character, say v-4.1a denote the first development line (rcs). The following version numbers have the form “v⟨n⟩⟨n⟩⟨n⟩” with ⟨n⟩ as a decimal number, e. g. v122 (subversion). The current development is under git and uses version numbers of the form ⟨d⟩.⟨d⟩[⟨a⟩][⟨d⟩]-g⟨x⟩, for example, v2.1-13-gd28a927; ⟨x⟩ stands for a hexadecimal number. In general, the age and therefore the sequence of the versions is

$$v\text{-}\langle d \rangle.\langle d \rangle[\langle a \rangle] < v\langle n \rangle\langle n \rangle\langle n \rangle < v\langle d \rangle.\langle d \rangle[\langle a \rangle][\text{-}\langle d \rangle]\text{-}g\langle x \rangle$$

It is possible to draw structured box charts by this package of macros which is described herewith. Through this article the package will be always called St<sub>u</sub>kT<sub>E</sub>X. It can generate the most important elements of a structured box chart like processing blocks, loops, mapping conventions for alternatives etc.<sup>1</sup>

Since version v-4.1a the mathematical symbols are loaded by  $\mathcal{AMS}$ -T<sub>E</sub>X. They extend the mathematical character set and make other representations of symbols sets (like  $\mathbb{N}$ ,  $\mathbb{Z}$  and  $\mathbb{R}$  for the natural, the integer and the real numbers) possible. Especially the symbol for the empty set ( $\emptyset$ ) has a more outstanding representation than the standard symbol (“ $\emptyset$ ”). Therefore it is the better representation in structured box charts.

Furthermore the idea to set names of variables in *italics* without generating the partly unpleasant distances is taken over from `oz.sty`.

The development of this macro package is still not finished. It was planned to draw the structured box charts by using the macros of `emlines2.sty` for eliminating the constraints given by L<sup>A</sup>T<sub>E</sub>X. – There are only predefined gradients. – This is done for the `\ifthenelse` in the versions v-4.1a and v-4.1b and for `\switch` in the version v-4.2a, but not for the systems, which do not support the corresponding `\special{...}`-commands. Nevertheless it can be attained by using the corresponding macros of `curves.sty`. Since version v-8.0a the package `pict2e` is supported. This package eliminates the above mentioned constraints by using the common drivers, so it is recommended to use the respective (see below) option permanently.

<sup>1</sup> Those who don’t like to code the diagrams by hand, can use for example the program Strukturator (<http://strukturizer.fisch.lu/>). By using this program one can draw the diagrams by mouse and export the result into a L<sup>A</sup>T<sub>E</sub>X-file.

Just so it is planned to extend structured box charts by comments as they are used in the book of Futschek ([Fut89]). This is also implemented in version v-8.0a.

Further plans for future are:

1. An `\otherwise`-branch at `\switch` (done in version v-4.2a).
2. The reimplementaion of the `declaration`-environment through the `list`-environment by [GMS94, Abs. 3.3.4] (done in version v-4.5a).
3. The adaption to L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> in the sense of packages (done in version v-4.0a).
4. The improvement of documentation in order to make parts of the algorithm more understandable.
5. The independence of `struktex.sty` of other `.sty`-files like e.g. `JHfMakro.sty` (done in version v-4.5a).
6. The complete implementation of the macros `\pVar`, `\pKey`, `\pFonts`, `\pTrue`, `\pFalse` and `\pBoolValue` (done before version v-7.0).
7. The complete internalization of commands, which only make sense in the environment `struktogramm`. Internalization means, that these commands are only defined in this environment. This is for compatibility of this package with other packages, e.g. with `ifthenelse.sty`. The internalization has been started in version v-4.4a.
8. The independence of the documentation of other `.sty`-files like `JHfMakro.sty` (done in version v-5.0).
9. an alternative representation of declarations as proposed by Rico Bolz
10. Reintroduction of the `make`-targets `dist-src` `dist-tar` and `dist-zip`.

The current state of the implementation is noted at suitable points.

### 3 Hints for maintenance and installation as well as driver file creating of this documentation

The package `struktex.sty` is belonging to consists of altogether two files:

```
LIESMICH,  
README,  
struktex.ins,  
struktex.dtx,  
struktex.de.pdf und  
struktex.en.pdf.
```

In order to generate on the one hand the documentation and on the other hand the `.sty`-file one has to proceed as follows:

First the file `struktex.ins` will be formatted e.g. with

```
tex struktex.ins
```

. This formatting run generates eleven further files. These are first of all the three .sty-files `struktex.sty`, `struktxf.sty` and `struktxp.sty`, that are used for `struktex.sty`. Furthermore these are the two files `struktex_test_0.nss` and `strukdoc.sty`, which are used for the generation of the hereby presented documentation. Then there are three test files `struktex_test_i.nss`,  $i = 1(2)3$  as well as the files `struktex.makemake` and `struktex.mk` (see section 8).

The common procedure to produce the documentation is<sup>2</sup>

```
pdflatex struktex.dtx
pdflatex struktex.dtx
makeindex -s gind.ist struktex.idx
pdflatex struktex.dtx
```

The result of this formatting run is the documentation in form of a .pdf-file, that can be manipulated the normal way. Further informations about the work with the integrated documentation one can find in [Mit01] and [MDB01].

To finish the installation, the file `struktex.sty` should be moved to a directory, where  $\text{\TeX}$  can find it, in a TDS conforming installation this is `.../tex/latex/struktex/` typical, analogously the documentation has to be moved to `.../doc/latex/struktex/`. If the installation process is done automatically (see section 8), the target directories follow that rule.

If one wants to carry out changes, the values of `\fileversion`, `\filedate` and `\docdate` should be also changed if needed. Furthermore one should take care that the audit report will be carried on by items in the form of

`\changes{<version>}{<date>}{<comment>}`

The version number of the particular change is given by `<version>`. The date is given by `<date>` and has the form `yy/mm/dd`. `<comment>` describes the particular change. It need not contain more than 64 characters. Therefore commands should'nt begin with `"\"` (*backslash*), but with the `"`"` (*accent*).

The following commands make up the driver of the documentation lying before.

```
1 <*driver>
2                                     % select the formatting language:
3 \expandafter\ifx\csname primarylanguage\endcsname\relax%
4 \def\primarylanguage{ngerman}%
5 \def\secondarylanguage{english}%
6 \else%
7 \def\secondarylanguage{ngerman}%
8 \fi
9
10 \documentclass[a4paper, \secondarylanguage,      % select the language
11               \primarylanguage]{ltxdoc}
12
13 \PassOptionsToPackage{obeyspaces}{url} % must be done before any package is
14                                     % loaded
15
16 \usepackage{babel}                  % for switching the documentation language
17 \usepackage{strukdoc}               % the style-file for formatting this
18                                     % documentation
19
```

---

<sup>2</sup>Generating the documentation is much easier with the `make` utility, see section 8.

```

20 \usepackage[pict2e, % <----- to produce finer results
21                               % visible under xdvi, alternatives are
22                               % curves or emlines2 (visible only under
23                               % ghostscript), leave out if not
24                               % available
25   verification,
26   outer, % <----- to set the position of the \ifthenelse
27                               % flags to the outer edges
28   debug,
29 ]
30 {struktex}
31 \GetFileInfo{struktex.sty}
32
33 \EnableCrossrefs
34 %\DisableCrossrefs % say \DisableCrossrefs if index is ready
35
36 %\RecordChanges % say \RecordChanges to gather update information
37
38 %\CodelineIndex % say \CodelineIndex to index entry code by line number
39
40 \OnlyDescription % say \OnlyDescription to omit the implementation details
41
42 \MakeShortVerb{\\} % |\foo| acts like \verb+\foo+
43
44 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
45 % to avoid underfull ... messages while formatting two/three columns
46 \hbadness=10000 \vbadness=10000
47
48 \typeout{\string\primarylanguage: \primarylanguage, \string\language: \the\language}
49
50 \def\languageNGerman{10} % depends on language.dat, put
51                          % \the\language here
52
53 \begin{document}
54 \makeatletter
55 \@ifundefined{selectlanguageEnglish}{}{\selectlanguage{english}}
56 \makeatother
57 \DocInput{struktex.dtx}
58 \end{document}
59 </driver>

```

## 4 The User interface

The `struktex.sty` will be included in a  $\text{\LaTeX}$ -document like every other `.sty`-file by *package*:

```
\usepackage[<options>]{struktex}
```

The following options are available:

1. `english`, `ngerman` oder `german`:

This option defines the language of defined values as `\sTrue`, default: `english`.

2. `emlines`, `curves`, or `pict2e`:

If you set one of these options, any ascent can be drawn in the structured box chart. You should use `emlines`, if you are working with the `emTeX`-package for DOS or OS/2 by E. Mattes, else you should use `pict2e`; `curves` is included just for compatibility. In all cases the required packages (`emline2.sty`, `curves.sty`, or `pict2e` resp.) will be loaded automatically; the default value is `pict2e`.

3. `verification`:

Only if this option is set, the command `\assert` is available.

4. `nofiller`:

Setting this option prohibits setting of  $\emptyset$  in alternatives.

5. `draft`, `final`:

These options serve as usual to differentiate between the draft and the final version of a structured box chart (see `\sProofOn`). While in the draft mode the user given size of the chart is denoted by the four bullets, the final version leaves out these markers; the default value is `final`.

6. `debug`:

Setting this option produces lines of the form `===> dbg <text>` in the .log-file.

7. `outer`:

Setting this option changes the position of flags in the `ifthenelse`-triangles from the mid to the left and right of the baselines of the triangles.

After loading the `.sty`-file there are different commands and environments, which enable the draw of structured box charts.

`\StrukTeX` First of all the logo `StrukTeX` producing command should be mentioned:

`\StrukTeX`

So in documentations one can refer to the style option given hereby.

## 4.1 Specific characters and text representation

<code>\nat</code>	Since sets of natural, whole, real and complex numbers ( $\mathbb{N}$ , $\mathbb{Z}$ , $\mathbb{R}$ and $\mathbb{C}$ ) occur often
<code>\integer</code>	in the Mathematics Mode they can be reached by the macros <code>\nat</code> , <code>\integer</code> ,
<code>\real</code>	<code>\real</code> and <code>\complex</code> . Similarly " $\emptyset$ ", which is generated by <code>\emptyset</code> , is the
<code>\complex</code>	more remarkable symbol for the empty statement than the standard symbol " $\emptyset$ ".
<code>\emptyset</code>	Other set symbols like $\mathbb{L}$ (for solution space) have to be generated by <code>\mathbb{L}</code> .
<code>\MathItalics</code>	One can influence the descriptions of variable names by these macros.
<code>\MathNormal</code>	

*NewValue = OldValue + Correction*

```
\MathNormal
\[
NewValue = OldValue + Correction
\]
```

und

$newValue = oldValue + correction$	<code>\MathItalics</code>
	<code>\[</code>
	<code>newValue = oldValue + correction</code>
	<code>\]</code>

## 4.2 Macros for representing variables, keywords and other specific details of programming

<code>\pVariable</code>	Structured box charts sometimes include code, that has to be programmed directly. For achieving a homogenous appearance the mentioned macros have been defined. They have been collected in a separate package <code>struktexp.sty</code> to be able to use them in another context. From version 122 on <code>struktexp.sty</code> is based on "url.sty" of Donald Arsenau. This package makes allows to pass verbatim texts as parameters to other macros. If this verbatim stuff contains blank spaces, which should be preserved, the user has to execute the command
<code>\pVar</code>	
<code>\pKeyword</code>	
<code>\pKey</code>	
<code>\pComment</code>	

`\PassOptionsToPackage{oobeyspaces}{url}`

before `url.sty` is loaded, that is in most of the cases before the command

`\usepackage{struktexp}`

Variable names are set by `\pVariable{<VariableName>}`. There `<VariableName>` is an identifier of a variable, whereby the underline "`_`", the commercial and "`&`" and the hat "`^`" are allowed to be parts of variables:

<code>cANormalVariable</code>	<code>\obeylines</code>
<code>c_a_normal_variable</code>	<code>\pVariable{cANormalVariable}</code>
<code>&amp;iAddressOfAVariable</code>	<code>\pVariable{c_a_normal_variable}</code>
<code>pPointerToAVariable^.sContent</code>	<code>\pVariable{&amp;iAddressOfAVariable}</code>
	<code>\pVariable{pPointerToAVariable^.sContent}</code>

Blanks are considered such, that whole statements can be written. For abbreviation it is allowed to use `\pVar`.

A keyword is set by `\pKeyword{<keyword>}` respectively. There `<keyword>` is a keyword in a programming language, whereby the underline "`_`" and the *hash* symbol "`#`" are allowed to be parts of keywords. Therewith the following can be set:

<code>begin</code>	<code>\obeylines</code>
<code>program</code>	<code>\pKeyword{begin}</code>
<code>#include</code>	<code>\renewcommand{\pLanguage}{Pascal}</code>
	<code>\pKeyword{program}</code>
	<code>\renewcommand{\pLanguage}{C}</code>
	<code>\pKeyword{#include}</code>

`\pKeyword` is also allowed to be abbreviated by `\pKey`. With that the source code

```
\renewcommand{\pLanguage}{Pascal}
\pKey{begin} \pExp{iVar := iVar + 1;} \pKey{end}
```

generates the following result as output:

```
begin iVar := iVar + 1; end
```

In a similar way `\pComment` is of representation purposes of comments. The argument is only allowed to consist of characters of the category *letter*. Characters, that start a comment, have to be written. `\pComment` can't be abbreviated. For instance

```
\pExp{a = sqrt(a);} \pComment{// Iteration}
```

results in the line

```
a = sqrt(a); // Iteration
```

`\pTrue` Boolean values play an important role in programming. There are given adequate values by `\pTrue` and `\pFalse`: `true` and `false`.  
`\pFalse`  
`\pFonts` The macro `\pFonts` is used for the choice of fonts for representation of variables, keywords and comments:  
`\pBoolValue`

```
\pFonts{\langle variablefont \rangle}{\langle keywordfont \rangle} {\langle commentfont \rangle}
```

The default values for the certain fonts are

- `\langle variablefont \rangle` as `\small\sffamily`,
- `\langle keywordfont \rangle` as `\small\sffamily\bfseries` and
- `\langle commentfont \rangle` as `\small\sffamily\slshape`.

With that the above line becomes

```
a = sqrt(a); // Iteration
```

Similarly the values of `\pTrue` and `\pFalse` can be redefined by the macro

```
\sBoolValue{\langle Yes-Value \rangle}{\langle No-Value \rangle}
```

So the lines

```
\renewcommand{\pLanguage}{Pascal}
\sBoolValue{\textit{yes}}{\textit{no}}
\pFalse = \pKey{not} \pTrue
```

result in the following:

```
no= not yes
```

`\sVar` The macros `\sVar` and `\sKey` are the same as the macros `pVar` and `pKey`.  
`\sKey` Here they are just described for compatibility reasons with former versions of  
`\sTrue` `struktex.sty`. The same rule shall apply to the macros `\sTrue` and `\sFalse`.  
`\sFalse`



### 4.3 The Macros for generating structured box charts

<code>struktogramm</code>	The environment
<code>\sProofOn</code>	
<code>\sProofOff</code>	<code>\begin{struktogramm}(\langle width \rangle, \langle height \rangle) [\langle titel \rangle]</code>
<code>\PositionNSS</code>	<code>...</code>
	<code>\end{struktogramm}</code>

generates space for a new box chart. Both the parameters provide the width and the height of the place, which is reserved for the structured box chart. Lengths etc. are described in millimeters. In doing so the actual value of `\unitlength` is unimportant. At the same time the width corresponds with the real width and the real height will be adjusted to the demands. If the given height doesn't match with the real demands, the structured box chart reaches into the surrounding text or there is empty space respectively. There is a switch `\sProofOn`, with which the stated dimensions of the structured box charts is given by four points to make corrections easier. `\sProofOff` similarly switches this help off. The title is for identification of structured box charts, if one wants to refer to this from another part, e.g. from a second box chart.

The structured box chart environment is based on the `picture` environment of  $\text{\LaTeX}$ . The unit of length `\unitlength`, which is often used in the `picture` environment, is not used in structured box charts. The unit of length is fixed by 1 mm for technical reasons. Furthermore all of length specifications have to be whole numbers. After drawing a structured box chart by  $\text{\StukTeX}$  `\unitlength` is of the same quantity as before. But it is redefined within a structured box chart and need not be changed there.

<code>\assign</code>	The main element of a structured box chart is a box, in which an operation is described. Such a box will be assigned by <code>\assign</code> . The syntax is the following:
----------------------	---

`\assign[\langle height \rangle]{\langle content \rangle},`

where the square brackets name an optional element as usual. The width and the height of the box will be adjusted automatically according to demands. But one can predefine the height of the box by the optional argument.

The *text* is normally set centered in the box. If the text is too long for that, then a (justified) paragraph is set.

#### Example 1

A simple structured box chart will be generated by the following instructions:

```
\sProofOn
\begin{struktogramm}(70,20)[1.\ trial]
  \assign{Root of  $\pi$ , calculation and output}
\end{struktogramm}
\sProofOff
```

These instructions lead to the following box chart, at which the user has to provide an appropriate positioning like in the basing `\picture` environment. Herewith the positioning is normally done by the `quote` environment. But one can also center the structured box chart by the `center` environment. The width of the box chart is given by 70mm, the

height by 12mm. An alternative is given by the `centernss` environment, that is described on page 21

At the same time the effect of `\sProofOn` and `\sProofOff` is shown, at which the too large size of structured box chart has to be taken notice of.

1. trial

Root of  $\pi$ , calculation and output

The meaning of the optional argument will be made clear by the following example:

### Example 2

The height of the box is given by:

```
\begin{center}
\begin{struktogramm}(70,20)
  \assign[20]{Root of $\pi$, calculation and output}
\end{struktogramm}
\end{center}
```

These instructions lead to the following structured box chart. In doing so it is to pay attention on the `struktogramm` environment, which has been centered by the `center` environment, at which the width of the structured box chart is again given by 70mm, but the height by 20mm this time.

Root of  $\pi$ , calculation and output

**declaration** The `declaration` environment is used for the description of variables or interfaces respectively. Its syntax is given by

```
\begin{declaration}[\langle title \rangle]
...
\end{declaration}
```

**\declarationtitle** The declaration of the title is optional. If the declaration is omitted, the standard title: ‘Providing Memory Space’ will be generated. If one wants to have another text, it will be provided globally by `\declarationtitle{\langle title \rangle}`. If one wants to generate a special title for a certain structured box chart, one has to declare it within square brackets.

**\description** Within the `declaration` environment the descriptions of the variables can be generated by

```
\descriptionindent
\descriptionwidth
\descriptionsep
\description{\langle variableName \rangle}{\langle variableDescription \rangle}
```

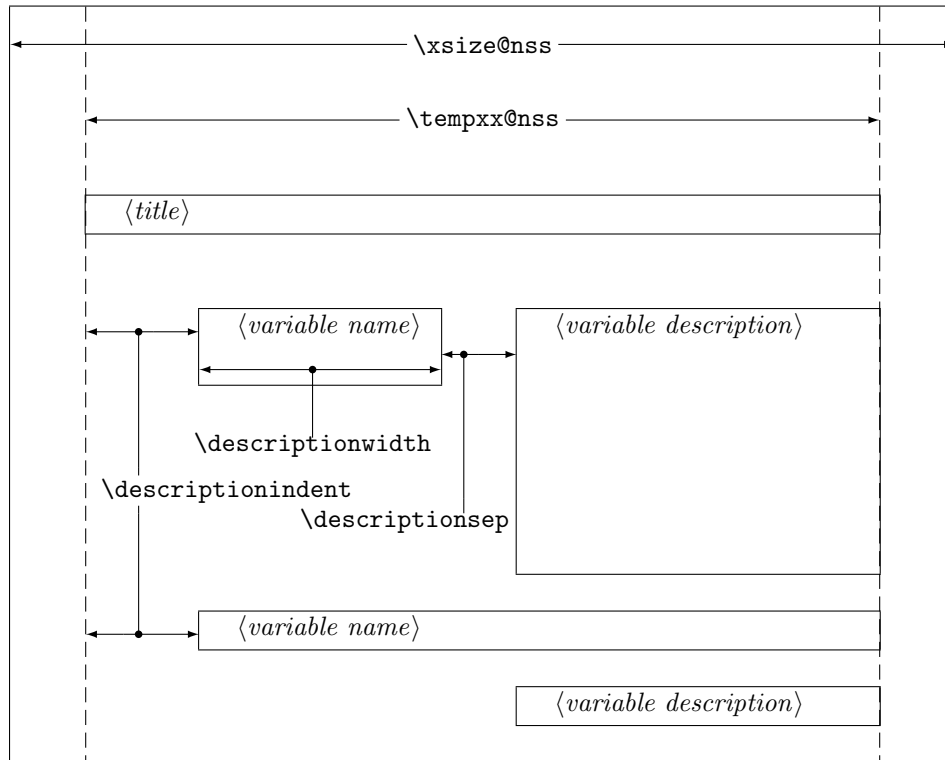


Figure 1: Construction of a Variable Description

In doing so one has to pay attention on the  $\langle variableName \rangle$ , that is not allowed to content a right square bracket `”]”`, because this macro has been defined by the `\item` macros. Square brackets have to be entered as `\lbracket` or `\rbracket` respectively.

The shape of a description can be controled by three parameters: `\descriptionindent`, `\descriptionwidth` and `\descriptionsep`. The meaning of the parameters can be taken from 1 (`\xsize@nss` and `\xin@nss` are internal sizes, that are given by `StylTeX`). The default values are the following:

```
\descriptionindent=1.5em
\descriptionwidth=40pt
\descriptionsep=\tabcolsep
```

The significance of `\descriptionwidth` is, that a variable name, which is shorter than `\descriptionwidth`, gets a description of the same height. Otherwise the description will be commenced in the next line.

### Example 3

First there will be described only one variable.

```
\begin{struktogramm}(95,20)
  \assign%
  {%
    \begin{declaration}
```

```

\description{\pVar{iVar}}{an \pKey{int} variable, which is
                    described here just for presentation of the
                    macro}
\end{declaration}
}
\end{struktogramm}

```

The corresponding structured box chart is the following, at which one has to pay attention, that there are no titles generated by the empty square brackets.

providing memory space: <i>iVar</i> {an <i>int</i> variable, which is described here just for presentation of the macro}
--

Now variables will be specified more precisely:

```

\begin{struktogramm}(95,50)
\assign{
\begin{declaration}[Parameter:]
\description{\pVar{iPar}}{an \pKey{int} parameter with the
                    meaning described here}
\end{declaration}
\begin{declaration}[local Variables:]
\description{\pVar{iVar}}{an \pKey{int} variable with the meaning
                    described here}
\description{\pVar{dVar}}{a \pKey{double} variable with the
                    meaning described here}
\end{declaration}
}
\end{struktogramm}

```

This results in:

Parameter: <i>iPar</i> {an <i>int</i> parameter with the meaning described here}
local Variables: <i>iVar</i> {an <i>int</i> variable with the meaning de- scribed here}
<i>dVar</i> {a <i>double</i> variable with the meaning described here}

Finally the global declaration of a title:

```

\def\declarationtitle{global variables}
\begin{struktogramm}(95,13)

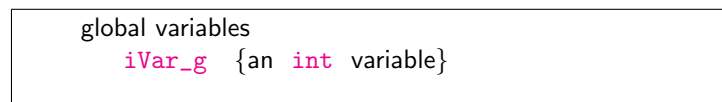
```

```

\assign{%
\begin{declaration}
\description{\pVar{iVar_g}}{an \pKey{int} variable}
\end{declaration}
}
\end{struktogramm}

```

This results in the following shape:



Here one has to notice the local realisation of the `\catcode` of the underline, which is necessary, if one wants to place an underline into an argument of macro. Although this local transfer is already realized at `\pVar` it doesn't suffice with the technique of macro expanding of  $\text{\TeX}$ .

`\sub`      The mapping conventions for jumps of subprograms and for exits of program  
`\return` look similar and are drawn by the following instructions:

```

\sub[⟨height⟩]{⟨text⟩}
\return[⟨height⟩]{⟨text⟩}

```

The parameters mean the same as at `\assign`. The next example shows how the mapping conventions are drawn.

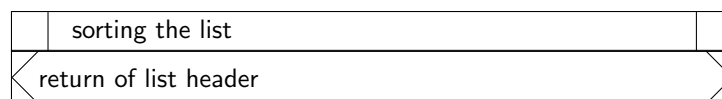
#### Example 4

```

\begin{struktogramm}(95,20)
\sub{sorting the list}
\return{return of list header}
\end{struktogramm}

```

These instructions lead to the following structured box chart:



`\while`      For representation of loop constructions there are three instructions available:  
`\whileend` `\while`, `\until` and `\forever`. The while loop is a repetition with preceding  
`\until` condition check (loop with a pre test). The until loop checks the condition at the  
`\untilend` end of the loop (loop with a post test). And the forever loop is a neverending  
`\forallin` loop, that can be left by `\exit`.  
`\forallinend`      `\while`[⟨width⟩]{⟨text⟩}{structured subbox chart}  
`\forever`      `\whileend`  
`\foreverend`      `\until`[⟨width⟩]{⟨text⟩}{structured subbox chart}

```

\untilend
\forever[⟨width⟩]⟨structured subbox chart⟩\foreverend
\exit[⟨height⟩]⟨text⟩

```

⟨width⟩ is the width of frame of the mapping convention and ⟨text⟩ is the conditioning text, that is written inside this frame. If the width is not given, the thickness of frame depends on the height of text. The text will be written left adjusted inside the frame. If there isn't given any text, there will be a thin frame.

A control structure which nowadays is provided by many programming languages is a loop known as `forall`-, `for ... in`-, or `foreach`-loop. This kind of loop can be seen as tail first loop but some people prefer the form of the endless loop with included text. For this case there is the control structure

```

\forallin[⟨width⟩]{⟨text⟩}⟨structured subbox chart⟩\forallin

```

Instead of ⟨structured subbox chart⟩ there might be written any instructions of `StruktTeX` (except `\openstrukt` and `\closestrukt`), which build up the box chart within the `\while` loop, the `\until` loop or the `\forever` loop.

For compatibility with further development of the `struktex.sty` of J. Dietel there are the macros `\dfr` and `\dfrend` with the same meaning as `\forever` and `\foreverend`.

The two following examples show use of `\while` and `\until` macros. `\forever` will be shown later.

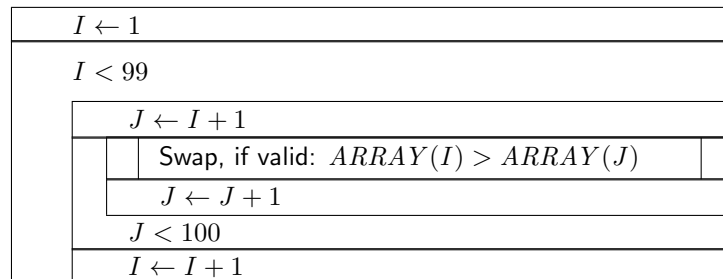
#### Example 5

```

\begin{struktogramm}(95,40)
  \assign{(I \gets 1)}
  \while[8]{(I < 99)}
    \assign{(J \gets I+1)}
    \until{(J < 100)}
      \sub{Swap, if valid: \(\ ARRAY(I) > ARRAY(J) \)}
      \assign{(J \gets J+1)}
    \untilend
  \assign{(I \gets I+1)}
\whileend
\end{struktogramm}

```

These instructions lead to the following structured box chart:



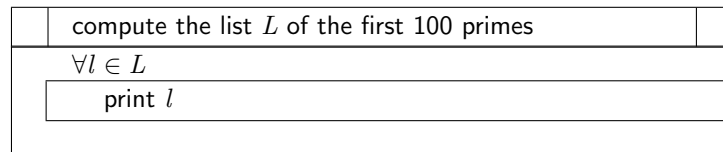
#### Example 6

```

\begin{struktogramm}(95, 25)
  \sub{compute the list \(\mathcal{L}\) of the first 100 primes}
  \forallin{\(\forall l \in \mathcal{L}\)}
    \assign{print \(\mathcal{L}\)}
  \forallinend
\end{struktogramm}

```

These instructions lead to the following structured box chart:



The `\exit` instruction only makes sense in connection with simple or multiple branches. Therefore it will be presented after the discussion of branches.

`\ifthenelse` For the representation of alternatives  $\text{\texttt{Strukt}}\text{\texttt{TeX}}$  provides mapping conventions for an If-Then-Else-block and a Case-construction for multiple alternatives. Since in the traditional `picture` environment of  $\text{\texttt{LaTeX}}$  only lines of certain gradients can be drawn, in both cases the user has to specify himself the angle, with which the necessary slanted lines shall be drawn. (Here is a little bit more ‘handy work’ required.)

If however the `curves.sty`, the `emlines2.sty` or the `pict2e.sty` is used, then the representation of lines with any gradient can be drawn.

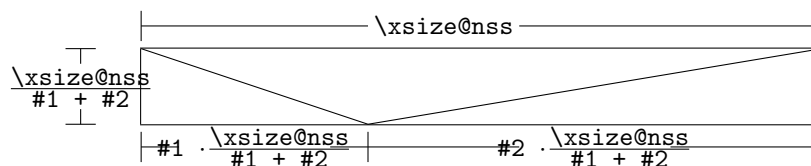
The If-Then-Else-command looks like:

```

\ifthenelse[ $\langle height \rangle$ ]{ $\langle left angle \rangle$ }{ $\langle right angle \rangle$ }
  { $\langle condition \rangle$ }{ $\langle left text \rangle$ }{ $\langle right text \rangle$ }
   $\langle structured subbox chart \rangle$ 
\change
   $\langle structured subbox chart \rangle$ 
\ifend

```

In the case of omitting the optional argument  $\langle height \rangle$   $\langle left angle \rangle$  and  $\langle right angle \rangle$  are numbers from 1 to 6. They specify the gradient of both the partitioning lines of the If-Then-Else-block (large number = small gradient). Larger values are put on 6, smaller values on 1. The precise characteristics of the gradients can be taken from the following picture. Thereby `\xsize@nss` is the width of the actual structured subbox chart. If the  $\langle height \rangle$  is given, then this value determines the height of the conditioning rectangle instead of the expression  $\frac{\text{\texttt{\xsize@nss}}}{\#1 + \#2}$ .



$\langle condition \rangle$  is set in the upper triangle built in the above way. The parameters  $\langle left\ text \rangle$  and  $\langle right\ text \rangle$  are set in the left or right lower triangle respectively. The conditioning text can be made up in its triangle box. From version v-5.3 on the conditioning text ...<sup>3</sup> Both the other texts should be short (e.g. yes/no or true/false), since they can't be made up and otherwise they stand out from their triangle box. For obtaining uniformity here the macros `\pTrue` and `\pFalse` should be used. Behind `\ifthenelse` the instructions for the left "structured subbox chart" are written and behind `\change` the instructions for the right "structured subbox chart" are written. If these two box charts have not the same length, then a box with  $\emptyset$  will be completioned. The If-Then-Else-element is finished by `\ifend`. In the following there are two examples for application.

#### Example 7

```
\begin{struktogramm}(95,32)
  \ifthenelse[12]{1}{2}
    {Flag for Output on Printer set ?}{\sTrue}{\sFalse}
    \assign[15]{Output directed to Printer}
  \change
    \assign{Output on Screen}
  \ifend
\end{struktogramm}
```

These instructions lead to the following structured box chart:

Flag for Output on Printer set ?	
true	false
Output directed to Printer	Output on Screen
	$\emptyset$

#### Example 8

```
\begin{struktogramm}(90,30)
  \ifthenelse{3}{4}
    {Output on Printer set ?}{\sTrue}{\sFalse}
    \assign[15]{Output on Printer diverted}
  \change
    \assign{Output on Screen}
  \ifend
\end{struktogramm}
```

These instructions lead to the following structured box chart:

---

<sup>3</sup>This extension is due to Daniel Hagedorn, whom I have to thank for his work.



Output on Printer set ?	
true	false
Output on Printer di- verted	Output on Screen
	∅

`\case`  
`\switch`  
`\caseend`

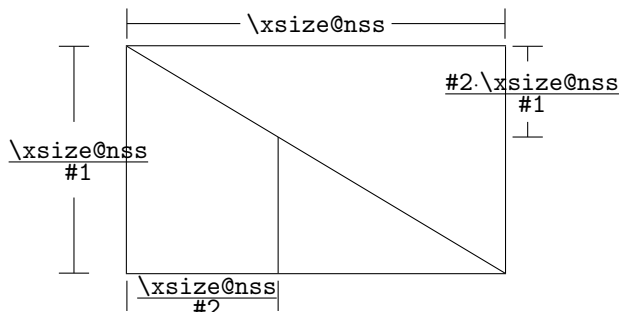
The Case-Construct has the following syntax:

```

\case[⟨height⟩]{⟨angle⟩}{⟨number of cases⟩}{⟨condition⟩}{⟨text of 1.
case⟩}}
  ⟨structured subbox chart⟩
\switch[⟨position⟩]{⟨text of 2. case⟩}
  ⟨structured subbox chart⟩
...
\switch[⟨position⟩]{⟨text of n. case⟩}
  ⟨structured subbox chart⟩
\caseend

```

If the *⟨height⟩* is not given, then the partitioning line of the mapping convention of case gets the gradient given by *⟨angle⟩* (those values mentioned at `\ifthenelse`). The text *⟨condition⟩* is set into the upper of the both triangles built by this line. The proportions are sketched below:



The second parameter *⟨number of cases⟩* specifies the number of cases, that have to be drawn. All structured subbox charts of the certain cases get the same width. The *⟨text of 1. case⟩* has to be given as a parameter of the `\case` instruction. All other cases are introduced by the `\switch` instruction. Behind the text the instructions for the proper structured subbox chart of certain case follow. The last case is finished by `\caseend`. A mapping convention of case with three cases is shown in the following example.

#### Example 9

```

\begin{struktogramm}(95,30)
\case{4}{3}{Signum(x)}{-1}
  \assign{$z$ \gets - \frac{1}{x}$}
\switch{0}

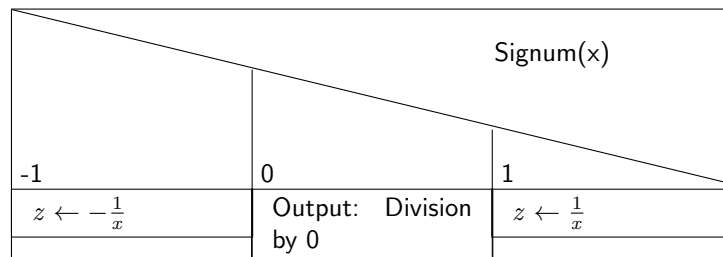
```

```

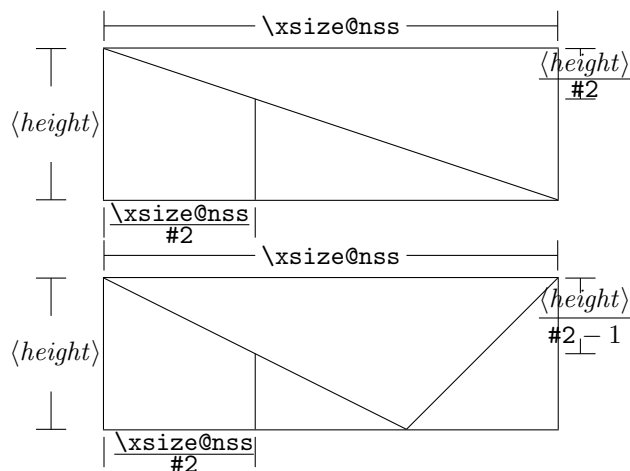
\assign{Output: Division by 0}
\switch{1}
\assign{$z \gets \frac{1}{x}$}
\caseend
\end{struktogramm}

```

These instructions lead to the following structured box chart:



The optional parameter [*height*] can be used if and only if one of the options “curves”, “emlines2” or “pict2e”, resp. is set; if this is not the case, the structured chart box may be scrambled up. The extension of the `\switch` instruction by [*height*] results in the following shape with a different gradient of a slanted line, which now is fixed by the height given by the optional parameter. If the value of the parameter *angle* is even, a straight line is drawn as before. If the value is odd, the last case is drawn as a special case as showed below.



### Example 10

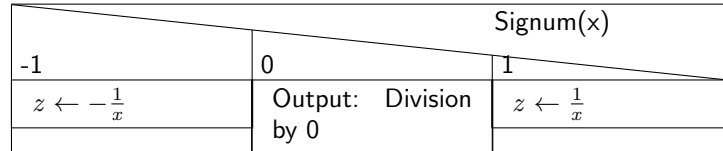
```

\begin{struktogramm}(95,30)
\case[10]{4}{3}{Signum(x)}{-1}
\assign{$z \gets - \frac{1}{x}$}
\switch{0}
\assign{Output: Division by 0}
\switch{1}
\assign{$z \gets \frac{1}{x}$}
\caseend

```

`\end{struktogramm}`

These instructions lead to the following structured box chart:

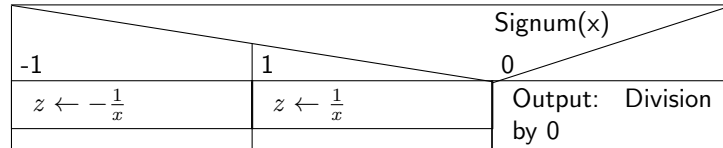


But if the first parameter is odd, then a default branch is drawn; the value for the default branch should be set flushed right.

#### Example 11

```
\begin{struktogramm}(95,30)
  \case[10]{5}{3}{Signum(x)}{-1}
    \assign{$z \gets - \frac{1}{x}$}
  \switch{1}
    \assign{$z \gets \frac{1}{x}$}
  \switch{0}
    \assign{Output: Division by 0}
  \caseend
\end{struktogramm}
```

These instructions lead to the following structured box chart:



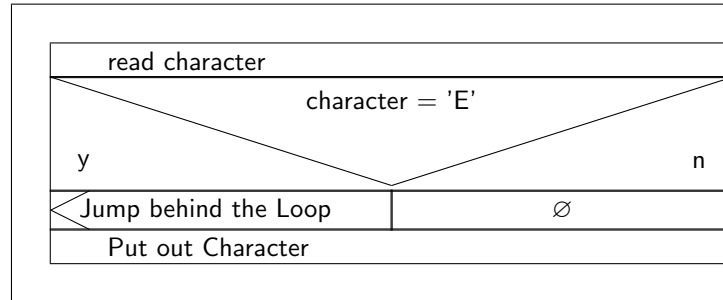
The following example shows, how one can exit a neverending loop by a simple branch. The example is transferable to a multiple branch without much effort.

#### Example 12

```
\begin{struktogramm}(95,40)
  \forever
    \assign{read character}
    \ifthenelse{3}{3}{character = 'E'}
      {y}{n}
    \exit{Jump behind the Loop}
  \change
  \ifend
  \assign{Put out Character}
\foreverend
```

`\end{struktogramm}`

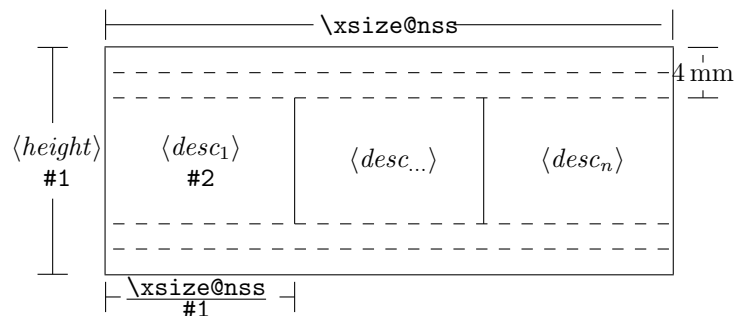
These instructions lead to the following structured box chart:



`\inparallel` Nowadays multicore processors or even better massive parallel processors are  
`\task` a common tool for executing programs. To use the features of these processors  
`\inparallelend` parallel algorithms should be developed and implemented. The `\inparallel` com-  
mand enables the representation of parallel processing in a program. The syntax  
is as follows:

```
\inparallel[⟨height of 1st task⟩]{⟨number of
parallel tasks⟩}{⟨description of 1st task⟩}
\task[⟨position⟩]{⟨description of 2nd task⟩}
...
\task[⟨position⟩]{⟨description of nth task⟩}
\inparallelend
```

The layout of the box is as follows (the macro parameters #1 and #2 refer to the parameters of `\inparallel`):



Note: the tasks are not allowed to get divided by `\assign` or so. If one needs some finer description of a task, this should be made outside of the current structured box chart.

**Example 13** (Application of `\inparallel`)

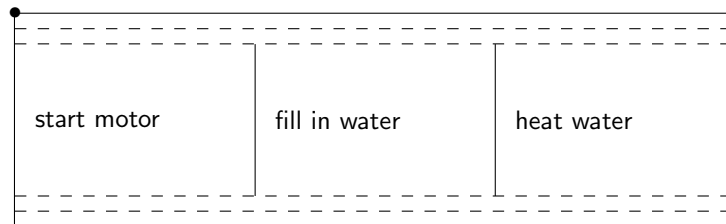
```
\begin{struktogramm}(95,40)
\inparallel[20]{3}{start motor}
```

```

\task{fill in water}
\task{heat water}
\inparallelel
\end{struktogramm}

```

These instructions produce the following structured box chart:



**centernss** If a structured box chart shall be represented centered, then the environment

```

\begin{centernss}
  \langle Struktogramm \rangle
\end{centernss}

```

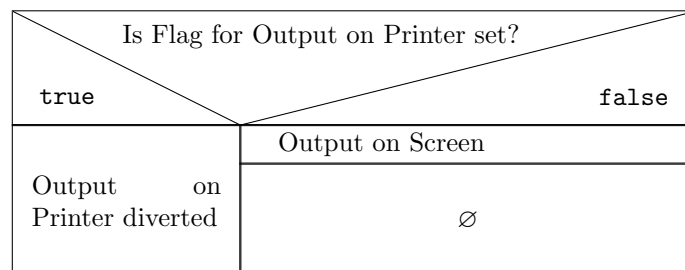
is used:

```

\begin{centernss}
\begin{struktogramm}(90,35)
  \ifthenelse{2}{4}
    {Is Flag for Output on Printer set?}{\sTrue}{\sFalse}%
    \assign[20]{Output on Printer diverted}
  \change
    \assign{Output on Screen}
  \ifend
\end{struktogramm}
\end{centernss}

```

This leads to the following:



**\CenterNssFile** In many cases structured box charts are recorded in particular files such, that they can be tested separately, if they are correct, or that they can be used in other connections. If they should be included centeredly, then one can not use the following construction:

```

\begin{center}
  \input{...}
\end{center}

```

since this way the whole text in structured box chart would be centered. To deal with this case in a simple and correct way the macro `\CenterNssFile` can be used. It is also defined in the style `centernssfile`. This requires, that the file containing the instructions for the structured box chart has the file name extension `.nss`. That is why the name of the file, that has to be tied in, *must* be stated without extension. If the file `struktex-test-0.nss` has the shape shown in paragraph 5, line 2–10 the instruction

```
\centernssfile{struktex-test-0}
```

leads to the following shape of the formatted text:

Text		
Signum(x)		
-1	0	1
$z \leftarrow -\frac{1}{x}$	Ausgabe: Divi- sion durch 0	$z \leftarrow \frac{1}{x}$

`\openstrukt`      These two macros are only preserved because of compatibility reasons with  
`\closestrukt`    previous versions of `StyTEX`. Their meaning is the same as `\struktogramm` and  
`\endstruktogramm`. The syntax is

```
\openstrukt{<width>}{<height>}
```

and

```
\closestrukt.
```

`\assert`      The macro `\assert` was introduced to support the verification of algorithms. It is active only if the option `verification` is set. It serves the purpose to assert the value of a variable at one point of the algorithm. The syntax corresponds to the syntax of `\assign`:

```
\assert[<height>]{<assertion>},
```

It's usage can be seen from the following:

```

\begin{struktogramm}(70,20)[Assertions in structured box charts]
  \assign{\(a\gets a^2\)}
  \assert{\(a\ge 0\)}
\end{struktogramm}
\sProofOff

```

The resulting structured box chart looks like

Assertions in structured box charts

$a \leftarrow a^2$
$a \geq 0$

## 5 Example file for including into the documentation

The following lines build up an example file, which is needed for the preparation of this documentation; there is only an german version.

```
60 (*example1)
61 \begin{struktogramm}(95,40)[Text]
62   \case[10]{3}{3}{Signum(x)}{-1}
63     \assign{(z \gets - \frac{1}{x})}
64   \switch{0}
65     \assign{Ausgabe: Division durch 0}
66   \switch[r]{1}
67     \assign{(z \gets \frac{1}{x})} \caseend
68 \end{struktogramm}
69 \end{example1}
```

## 6 Some example files

### 6.1 Example file for testing purposes of the macros of **struk-tex.sty** without any optional packages

The following lines build up a model file, that can be used for testing the macros.

```
70 (*example2)
71 \documentclass[draft]{article}
72 \usepackage{struktex}
73
74 \begin{document}
75
76 \begin{struktogramm}(90,137)
77   \assign%
78   {
79     \begin{declaration}[]
80       \description{(a, b, c)}{three variables which are to be sorted}
81       \description{(tmp)}{temporary variable for the circular swap}
82     \end{declaration}
83   }
84   \ifthenelse{1}{2}{(a \le c)}{j}{n}
85   \change
86   \assign{(tmp \gets a)}
87   \assign{(a \gets c)}
88   \assign{(c \gets tmp)}
89   \ifend
90   \ifthenelse{2}{1}{(a \le b)}{j}{n}
91   \ifthenelse{1}{1}{(b \le c)}{j}{n}
```

```

92   \change
93   \assign{\(tmp\gets c\)}
94   \assign{\(c\gets b\)}
95   \assign{\(b\gets tmp\)}
96   \ifend
97   \change
98   \assign{\(tmp\gets a\)}
99   \assign{\(a\gets b\)}
100  \assign{\(b\gets tmp\)}
101  \ifend
102 \end{struktogramm}
103
104 \end{document}
105 \end{example2}

```

## 6.2 Example file for testing purposes of the macros of **struktex.sty** with the package **pict2e.sty**

The following lines build up a template file, that can be used for testing the macros.

```

106 \begin{example3}
107 \documentclass{article}
108 \usepackage[pict2e, verification]{struktex}
109
110 \begin{document}
111 \def\StruktBoxHeight{7}
112 %\sProofOn{}
113 \begin{struktogramm}(90,137)
114   \assign%
115   {
116     \begin{declaration}[]
117       \description{\(a, b, c\)}{three variables which are to be sorted}
118       \description{\(tmp\)}{temporary variable for the circular swap}
119     \end{declaration}
120   }
121   \assert[\StruktBoxHeight]{\sTrue}
122   \ifthenelse[\StruktBoxHeight]{1}{2}{\(\(a\le c\)\){j}{n}}
123     \assert[\StruktBoxHeight]{\(\(a\le c\)\)}
124   \change
125     \assert[\StruktBoxHeight]{\(\(a>c\)\)}
126     \assign[\StruktBoxHeight]{\(\(tmp\gets a\)\)}
127     \assign[\StruktBoxHeight]{\(\(a\gets c\)\)}
128     \assign[\StruktBoxHeight]{\(\(c\gets tmp\)\)}
129     \assert[\StruktBoxHeight]{\(\(a<c\)\)}
130   \ifend
131   \assert[\StruktBoxHeight]{\(\(a\le c\)\)}
132   \ifthenelse[\StruktBoxHeight]{2}{1}{\(\(a\le b\)\){j}{n}}
133     \assert[\StruktBoxHeight]{\(\(a\le b \wedge a\le c\)\)}
134     \ifthenelse[\StruktBoxHeight]{1}{1}{\(\(b\le c\)\){j}{n}}
135     \assert[\StruktBoxHeight]{\(\(a\le b \le c\)\)}
136   \change
137     \assert[\StruktBoxHeight]{\(\(a \le c < b\)\)}
138     \assign[\StruktBoxHeight]{\(\(tmp\gets c\)\)}
139     \assign[\StruktBoxHeight]{\(\(c\gets b\)\)}

```



```

140         \assign[\StruktBoxHeight]{\b\gets tmp\)}
141         \assert[\StruktBoxHeight]{\a\le b<c\)}
142     \ifend
143 \change
144     \assert[\StruktBoxHeight]{\b < a\le c\)}
145     \assign[\StruktBoxHeight]{\tmp\gets a\)}
146     \assign[\StruktBoxHeight]{\a\gets b\)}
147     \assign[\StruktBoxHeight]{\b\gets tmp\)}
148     \assert[\StruktBoxHeight]{\a<b\le c\)}
149 \ifend
150 \assert[\StruktBoxHeight]{\a\le b \le c\)}
151 \end{struktoqramm}
152
153 \end{document}
154 \example3

```

### 6.3 Example file for testing the macros of `struktxp.sty`

The following lines build a sample file, which can be used for testing the macros of `struktxp.sty`. For testing one should delete the comment characters before the line `\usepackage[T1]{fontenc}`.

```

155 \example4
156 \documentclass[english]{article}
157
158 \usepackage{babel}
159 \usepackage{struktex}
160
161 \nofiles
162
163 \begin{document}
164
165 \pLanguage{Pascal}
166 \section*{Default values (Pascal):}
167
168 {\obeylines
169 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
170 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
171 in math mode: \(\pVar{a}+\pVar{iV_g}\)
172 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
173 }
174
175 \paragraph{After changing the boolean values with}
176 \verb-\pBoolValue{yes}{no}-:
177
178 {\obeylines
179 \pBoolValue{yes}{no}
180 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
181 }
182
183 \paragraph{after changing the fonts with}
184 \verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:
185
186 {\obeylines

```

```

187 \pFonts{\itshape}{\sffamily\bfseries}{\}
188 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
189 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
190 in math mode: \(\pVar{a}+\pVar{iV_g}\)
191 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
192 }
193
194 \paragraph{after changing the fonts with}
195 \verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:
196
197 {\obeylines
198 \pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
199 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
200 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
201 in math mode: \(\pVar{a}+\pVar{iV_g}\)
202 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
203 }
204
205 \paragraph{after changing the fonts with}
206 \verb-\pFonts{\itshape}{\bfseries\itshape}{\}-:
207
208 {\obeylines
209 \pFonts{\itshape}{\bfseries\itshape}{\}
210 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
211 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
212 in math mode: \(\pVar{a}+\pVar{iV_g}\)
213 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
214
215 \vspace{15pt}
216 Without \textit{italic correction}:
217 M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
218 }
219
220 \pLanguage{C}
221 \pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
222 \section*{Default values (C):}
223
224 {\obeylines
225 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
226 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
227 in math mode: \(\pVar{a}+\pVar{iV_g}\)
228 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
229 }
230
231 \paragraph{After changing the boolean values with}
232 \verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:
233
234 {\obeylines
235 \pBoolValue{\texttt{yes}}{\texttt{no}}
236 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
237 }
238
239 \paragraph{after changing the fonts with}
240 \verb-\pFonts{\itshape}{\sffamily\bfseries}{\}-:

```

```

241
242 {\obeylines
243 \pFonts{\itshape}{\sffamily\bfseries}{\}
244 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
245 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
246 in math mode: \(\pVar{a}+\pVar{iV_g}\)
247 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
248 }
249
250 \paragraph{after changing the fonts with}
251 \verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:
252
253 {\obeylines
254 \pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
255 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
256 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
257 in math mode: \(\pVar{a}+\pVar{iV_g}\)
258 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
259 }
260
261 \paragraph{after changing the fonts with}
262 \verb-\pFonts{\itshape}{\bfseries\itshape}{\}-:
263
264 {\obeylines
265 \pFonts{\itshape}{\bfseries\itshape}{\}
266 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
267 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
268 in math mode: \(\pVar{a}+\pVar{iV_g}\)
269 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
270
271 \vspace{15pt}
272 Without \textit{italic correction}:
273     M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
274 }
275
276 \pLanguage{Java}
277 \pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
278 \section*{Default values (Java):}
279
280 {\obeylines
281 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
282 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
283 in math mode: \(\pVar{a}+\pVar{iV_g}\)
284 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
285 }
286
287 \paragraph{After changing the boolean values with}
288 \verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:
289
290 {\obeylines
291 \pBoolValue{\texttt{yes}}{\texttt{no}}
292 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
293 }
294

```

```

295 \paragraph{after changing the fonts with}
296 \verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:
297
298 {\obeylines
299 \pFonts{\itshape}{\sffamily\bfseries}{}
300 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
301 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
302 in math mode: \(\pVar{a}+\pVar{iV_g}\)
303 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
304 }
305
306 \paragraph{after changing the fonts with}
307 \verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:
308
309 {\obeylines
310 \pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
311 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
312 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
313 in math mode: \(\pVar{a}+\pVar{iV_g}\)
314 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
315 }
316
317 \paragraph{after changing the fonts with}
318 \verb-\pFonts{\itshape}{\bfseries\itshape}{}-:
319
320 {\obeylines
321 \pFonts{\itshape}{\bfseries\itshape}{}
322 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
323 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
324 in math mode: \(\pVar{a}+\pVar{iV_g}\)
325 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
326
327 \vspace{15pt}
328 Without \textit{italic correction}:
329 M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
330 }
331
332 \pLanguage{Python}
333 \pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
334 \section*{Default values (Python):}
335
336 {\obeylines
337 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
338 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
339 in math mode: \(\pVar{a}+\pVar{iV_g}\)
340 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
341 }
342
343 \paragraph{After changing the boolean values with}
344 \verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:
345
346 {\obeylines
347 \pBoolValue{\texttt{yes}}{\texttt{no}}
348 boolean values: \sTrue, \sFalse, \pTrue, \pFalse

```

```

349 }
350
351 \paragraph{after changing the fonts with}
352 \verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:
353
354 {\obeylines
355 \pFonts{\itshape}{\sffamily\bfseries}{\sffamily\bfseries}{\sffamily\bfseries}
356 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
357 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
358 in math mode: \(\pVar{a}+\pVar{iV_g}\)
359 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
360 }
361
362 \paragraph{after changing the fonts with}
363 \verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:
364
365 {\obeylines
366 \pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}{\ttfamily\slshape}
367 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
368 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
369 in math mode: \(\pVar{a}+\pVar{iV_g}\)
370 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
371 }
372
373 \paragraph{after changing the fonts with}
374 \verb-\pFonts{\itshape}{\bfseries\itshape}{\bfseries\itshape}{\bfseries\itshape}-:
375
376 {\obeylines
377 \pFonts{\itshape}{\bfseries\itshape}{\bfseries\itshape}{\bfseries\itshape}
378 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
379 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
380 in math mode: \(\pVar{a}+\pVar{iV_g}\)
381 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
382
383 \vspace{15pt}
384 Without \textit{italic correction}:
385 M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
386 }
387
388 \end{document}
389 %%
390 %% End of file 'struktex-test-2.tex'.
391 \end{example4}

```

## 6.4 Example file for testing the macros of struktexp.sty

```

392 \documentclass{article}
393
394 \usepackage{struktexp, struktexp}
395
396 \makeatletter
397 \newlength{\fdesc@len}
398 \newcommand{\fdesc@label}[1]{%

```

```

400 {%
401     \settowidth{\fdesc@len}{\fdesc@font #1}%
402     \advance\hsize by -2em
403     \ifdim\fdesc@len>\hsize%                % term > labelwidth
404         \parbox[b]{\hsize}%
405         {%
406             \fdesc@font #1%
407         }\\%
408     \else%                                % term < labelwidth
409     \ifdim\fdesc@len>\labelwidth%          % term > labelwidth
410         \parbox[b]{\labelwidth}%
411         {%
412             \makebox[Opt][l]{\fdesc@font #1}\\%
413         }%
414     \else%                                % term < labelwidth
415         {\fdesc@font #1}%
416     \fi\fi%
417     \hfil\relax%
418 }
419 \newenvironment{fdescription}[1][\tt]%
420 {%
421     \def\fdesc@font{#1}
422     \begin{quote}%
423     \begin{list}{}%
424     {%
425         \renewcommand{\makelabel}{\fdesc@label}%
426         \setlength{\labelwidth}{120pt}%
427         \setlength{\leftmargin}{\labelwidth}%
428         \addtolength{\leftmargin}{\labelsep}%
429     }%
430 }%
431 {%
432     \end{list}%
433     \end{quote}%
434 }
435 \makeatother
436
437 \pLanguage{Java}
438
439 \begin{document}
440
441 \begin{fdescription}
442 \item[\index{Methoden}>drawImage(Image img,
443                                     int dx1,
444                                     int dy1,
445                                     int dx2,
446                                     int dy2,
447                                     int sx1,
448                                     int sy1,
449                                     int sx2,
450                                     int sy2,
451                                     ImageObserver observer)=%
452 \Expr{\pKey{public} \pKey{abstract} \pKey{boolean} drawImage(Image img,
453                                     \pKey{int} dx1,

```

```

454             \pKey{int} dy1,
455             \pKey{int} dx2,
456             \pKey{int} dy2,
457             \pKey{int} sx1,
458             \pKey{int} sy1,
459             \pKey{int} sx2,
460             \pKey{int} sy2,
461             ImageObserver observer)}}%
462     \pExp{public abstract boolean drawImage(Image img, int dx1, int
463         dy1, int dx1, int dy2, int sx1, int sy1, int sx2, int sy2,
464         ImageObserver observer)}}%
465     \ldots
466 \end{fdescription}
467 \end{document}
468 %%
469 %% End of file 'struktex-test-5.tex'.
470 \</example5>

```

## 7 Macros for generating the documentation of the `struktex.sty`

To simplify the formatting of the documentation some macros are used, which are collected in a particular `.sty` file. An essential part is based on a modification of the `newtheorem` environment out of `latex.sty` for distinguishing examples. The implementation of abbreviations has been proposed in [Neu96].

Therefore some instructions of `verbatim.sty` have been adopted and modified, so that writing and reading by the `docstrip` package works. Finally an idea of Tobias Oetiker out of `layout.sty` also has been used, which has been developed in connection with "`lshort2e.tex` – The not so short introduction to LaTeX2e".

```

471 \<strukdoc>
472 \RequirePackage{ifpdf}
473 \newif\ifcolor \IfFileExists{color.sty}{\colortrue}{}
474 \ifpdf \RequirePackage{colorlinks}{hyperref}\else
475     \def\href#1{\texttt}\fi
476 \ifcolor \RequirePackage{color}\fi
477 \RequirePackage{nameref}
478 \RequirePackage{url}
479 \renewcommand\ref{\protect\T@ref}
480 \renewcommand\pageref{\protect\T@pageref}
481 \@ifundefined{zB}{}{\endinput}
482 \providecommand\pparg[2]{%
483     {\ttfamily(\meta{#1},\meta{#2}{\ttfamily})}}
484 \providecommand\envb[1]{%
485     {\ttfamily\char'\begin\char'\{#1\char'\}}}
486 \providecommand\enve[1]{%
487     {\ttfamily\char'\end\char'\{#1\char'\}}}
488 \newcommand{\zBspace}{z.\,B.}
489 \let\zB=\zBspace
490 \newcommand{\dhspace}{d.\,h.}
491 \let\dh=\dhspace
492 \let\foreign=\textit
493 \newcommand\Abb[1]{Abbildung~\ref{#1}}
494 \def\newexample#1{%

```

```

495 \ifnextchar[{\@oexmpl{#1}}{\@nexmpl{#1}}
496 \def\@nexmpl#1#2{%
497 \ifnextchar[{\@xnexmpl{#1}{#2}}{\@ynexmpl{#1}{#2}}}
498 \def\@xnexmpl#1#2[#3]{%
499 \expandafter\@ifdefinable\csname #1\endcsname
500 {\@definecounter{#1}\@newctr{#1}[#3]%
501 \expandafter\xdef\csname the#1\endcsname{%
502 \expandafter\noexpand\csname the#3\endcsname \@exmplcountersep
503 \@exmplcounter{#1}}%
504 \global\@namedef{#1}{\@exmpl{#1}{#2}}%
505 \global\@namedef{end#1}{\@endexample}}}
506 \def\@ynexmpl#1#2{%
507 \expandafter\@ifdefinable\csname #1\endcsname
508 {\@definecounter{#1}%
509 \expandafter\xdef\csname the#1\endcsname{\@exmplcounter{#1}}%
510 \global\@namedef{#1}{\@exmpl{#1}{#2}}%
511 \global\@namedef{end#1}{\@endexample}}}
512 \def\@oexmpl#1[#2]#3{%
513 \@ifundefined{c@#2}{\@nocounterr{#2}}%
514 {\expandafter\@ifdefinable\csname #1\endcsname
515 {\global\@namedef{the#1}{\@nameuse{the#2}}}%
516 \global\@namedef{#1}{\@exmpl{#2}{#3}}%
517 \global\@namedef{end#1}{\@endexample}}}}
518 \def\@exmpl#1#2{%
519 \refstepcounter{#1}%
520 \ifnextchar[{\@yexmpl{#1}{#2}}{\@xexmpl{#1}{#2}}}
521 \def\@xexmpl#1#2{%
522 \@beginexample{#2}{\csname the#1\endcsname}\ignorespaces}
523 \def\@yexmpl#1#2[#3]{%
524 \@opargbeginexample{#2}{\csname the#1\endcsname}{#3}\ignorespaces}
525 \def\@exmplcounter#1{\noexpand\arabic{#1}}
526 \def\@exmplcountersep{.}
527 \def\@beginexample#1#2{%
528 \@nbreaktrue\list{}{\setlength{\rightmargin}{\leftmargin}}%
529 \item[{\bfseries #1\ #2}]{\mbox{}\\sf}
530 \def\@opargbeginexample#1#2#3{%
531 \@nbreaktrue\list{}{\setlength{\rightmargin}{\leftmargin}}%
532 \item[{\bfseries #1\ #2}]{\mbox{} ( #3) \mbox{}\\sf}
533 \def\@endexample{\endlist}
534
535 \newexample{tExample}{\ifnum\language=\languageNGerman Beispiel\else Example\fi}
536
537 \newwrite\struktex@out
538 \newenvironment{example}%
539 {\begingroup% Lets keep the changes local
540 \bsphack
541 \immediate\openout \struktex@out \jobname.tmp
542 \let\do\@makeother\dospecials\catcode'\^M\active
543 \def\verbatim@processline{%
544 \immediate\write\struktex@out{\the\verbatim@line}}%
545 \verbatim@start}%
546 {\immediate\closeout\struktex@out\esphack\endgroup%
547 %
548 % And here comes the part of Tobias Oetiker

```



```

549 %
550 \par\small\addvspace{3ex plus 1ex}\vskip -\parskip
551 \noindent
552 \makebox[0.45\linewidth][l]{%
553 \begin{minipage}[t]{0.45\linewidth}
554   \vspace*{-2ex}
555   \setlength{\parindent}{0pt}
556   \setlength{\parskip}{1ex plus 0.4ex minus 0.2ex}
557   \begin{trivlist}
558     \item\input{\jobname.tmp}
559   \end{trivlist}
560 \end{minipage}}%
561 \hfill%
562 \makebox[0.5\linewidth][l]{%
563 \begin{minipage}[t]{0.50\linewidth}
564   \vspace*{-1ex}
565   \verbatiminput{\jobname.tmp}
566 \end{minipage}}
567 \par\addvspace{3ex plus 1ex}\vskip -\parskip
568 }
569
570 \newtoks\verbatim@line
571 \def\verbatim@startline{\verbatim@line{}}
572 \def\verbatim@addtoline#1{%
573   \verbatim@line\expandafter{\the\verbatim@line#1}}
574 \def\verbatim@processline{\the\verbatim@line\par}
575 \def\verbatim@finish{\ifcat$\the\verbatim@line$\else
576   \verbatim@processline\fi}
577
578 \def\verbatim@write#1{%
579   \@bsphack
580   \immediate\openout \struktex@out #1
581   \let\do\@makeother\dospecials
582   \catcode'\^M\active \catcode'\^^I=12
583   \def\verbatim@processline{%
584     \immediate\write\struktex@out
585       {\the\verbatim@line}}%
586   \verbatim@start}
587 \def\endverbatim@write{%
588   \immediate\closeout\struktex@out
589   \@esphack}
590
591 \ifundefined{vrb@catcodes}%
592   {\def\vrb@catcodes{%
593     \catcode'\!12\catcode'\[12\catcode'\]12}}{-}
594 \begingroup
595 \vrb@catcodes
596 \lccode'\!='\ \lccode'\[='\ \lccode'\]='\
597 \catcode'\~= \active \lccode'\~= '\^^M
598 \lccode'\C='C
599 \lowercase{\endgroup
600   \def\verbatim@start#1{%
601     \verbatim@startline
602     \if\noexpand#1\noexpand~%

```

```

603         \let\next\verbatim@
604     \else \def\next{\verbatim@#1}\fi
605     \next}%
606 \def\verbatim@#1~{\verbatim@@#1!end\@nil}%
607 \def\verbatim@@#1!end{%
608     \verbatim@addtoline{#1}%
609     \futurelet\next\verbatim@@@}%
610 \def\verbatim@@@#1\@nil{%
611     \ifx\next\@nil
612         \verbatim@processline
613         \verbatim@startline
614         \let\next\verbatim@
615     \else
616         \def\@tempa##1!end\@nil{##1}%
617         \@temptokena{!end}%
618         \def\next{\expandafter\verbatim@test\@tempa#1\@nil~}%
619     \fi \next}%
620 \def\verbatim@test#1{%
621     \let\next\verbatim@test
622     \if\noexpand#1\noexpand~%
623         \expandafter\verbatim@addtoline
624         \expandafter{\the\@temptokena}%
625         \verbatim@processline
626         \verbatim@startline
627         \let\next\verbatim@
628     \else \if\noexpand#1
629         \@temptokena\expandafter{\the\@temptokena#1}%
630     \else \if\noexpand#1\noexpand[%
631         \let\@tempc\@empty
632         \let\next\verbatim@testend
633     \else
634         \expandafter\verbatim@addtoline
635         \expandafter{\the\@temptokena}%
636         \def\next{\verbatim@#1}%
637     \fi\fi\fi
638     \next}%
639 \def\verbatim@testend#1{%
640     \if\noexpand#1\noexpand~%
641         \expandafter\verbatim@addtoline
642         \expandafter{\the\@temptokena[]}%
643         \expandafter\verbatim@addtoline
644         \expandafter{\@tempc}%
645         \verbatim@processline
646         \verbatim@startline
647         \let\next\verbatim@
648     \else\if\noexpand#1\noexpand[%
649         \let\next\verbatim@@testend
650     \else\if\noexpand#1\noexpand!%
651         \expandafter\verbatim@addtoline
652         \expandafter{\the\@temptokena[]}%
653         \expandafter\verbatim@addtoline
654         \expandafter{\@tempc}%
655         \def\next{\verbatim@!}%
656     \else \expandafter\def\expandafter\@tempc\expandafter

```

```

657         {\@tempc#1}\fi\fi\fi
658     \next}%
659 \def\verbatim@testend{%
660     \ifx\@tempc\@currenvir
661         \verbatim@finish
662         \edef\next{\noexpand\end{\@currenvir}%
663             \noexpand\verbatim@rescan{\@currenvir}}}%
664     \else
665         \expandafter\verbatim@addtoline
666         \expandafter{\the\@temptokena}%
667         \expandafter\verbatim@addtoline
668         \expandafter{\@tempc}}%
669     \let\next\verbatim@
670 \fi
671 \next}%
672 \def\verbatim@rescan#1#2~{\if\noexpand~\noexpand#2~\else
673     \@warning{Characters dropped after '\string\end{#1}'}\fi}}
674
675 \newread\verbatim@in@stream
676 \def\verbatim@readfile#1{%
677     \verbatim@startline
678     \openin\verbatim@in@stream #1\relax
679     \ifeof\verbatim@in@stream
680         \typeout{No file #1.}%
681     \else
682         \@addtofilelist{#1}%
683         \ProvidesFile{#1}[(verbatim)]%
684         \expandafter\endlinechar\expandafter\m@ne
685         \expandafter\verbatim@read@file
686         \expandafter\endlinechar\the\endlinechar\relax
687         \closein\verbatim@in@stream
688     \fi
689     \verbatim@finish
690 }
691 \def\verbatim@read@file{%
692     \read\verbatim@in@stream to\next
693     \ifeof\verbatim@in@stream
694     \else
695         \expandafter\verbatim@addtoline\expandafter{\expandafter\check@percent\next}%
696         \verbatim@processline
697         \verbatim@startline
698         \expandafter\verbatim@read@file
699     \fi
700 }
701 \def\verbatiminput{\begingroup\MacroFont
702     \@ifstar{\verbatim@input\relax}%
703     {\verbatim@input{\frenchspacing\@vobeyspaces}}}
704 \def\verbatim@input#1#2{%
705     \IfFileExists {#2}{\@verbatim #1\relax
706         \verbatim@readfile{\@filef@und}\endtrivlist\endgroup\@doendpe}%
707     {\typeout {No file #2.}\endgroup}}
708 \</strukdoc>

```

## 8 Makefile for the automatized generation of the documentation and the tests of the **struktex.sty**

```
709 (*makefile)
710 #-----
711 # Purpose: generation of the documentation of the struktex package
712 # Notice:  this file can be used only with dmake and the option "-B";
713 #           this option lets dmake interpret the leading spaces as
714 #           distinguishing characters for commands in the make rules.
715 #
716 # Rules:
717 #       - all-de:      generate all the files and the (basic) german
718 #                       documentation
719 #       - all-en:      generate all the files and the (basic) english
720 #                       documentation
721 #       - test:        format the examples
722 #       - history:     generate the documentation with revision
723 #                       history
724 #       - develop-de:  generate the german documentation with revision
725 #                       history and source code
726 #       - develop-en:  generate the english documentation with
727 #                       revision history and source code
728 #       - realclean
729 #       - clean
730 #       - clean-example
731 #
732 # Author:   Jobst Hoffmann, Fachhochschule Aachen, Standort Juelich
733 # Date:     2017/06/06
734 #-----
735
736 # The texmf-directory, where to install new stuff (see texmf.cnf)
737 # If you don't know what to do, search for directory texmf at /usr.
738 # With teTeX and linux often one of following is used:
739 #INSTALLTEXMF=/usr/TeX/texmf
740 #INSTALLTEXMF=/usr/local/TeX/texmf
741 #INSTALLTEXMF=/usr/share/texmf
742 #INSTALLTEXMF=/usr/local/share/texmf
743 # user tree:
744 #INSTALLTEXMF=$(HOME)/texmf
745 # Try to use user's tree known by kpsewhich:
746 INSTALLTEXMF='kpsewhich --expand-var '$$TEXMFHOME''
747 # Try to use the local tree known by kpsewhich:
748 #INSTALLTEXMF='kpsewhich --expand-var '$$TEXMFLOCAL''
749 # But you may set INSTALLTEXMF to every directory you want.
750 # Use following, if you only want to test the installation:
751 #INSTALLTEXMF=/tmp/texmf
752
753 # If texhash must run after installation, you can invoke this:
754 TEXHASH=texhash
755
756 ##### Edit following only, if you want to change defaults!
757
758 # The directory, where to install *.cls and *.sty
```

```

759 CLSDIR=$(INSTALLTEXMF)/tex/latex/$(PACKAGE)
760
761 # The directory, where to install documentation
762 DOCDIR=$(INSTALLTEXMF)/doc/latex/$(PACKAGE)
763
764 # The directory, where to install the sources
765 SRCDIR=$(INSTALLTEXMF)/source/latex/$(PACKAGE)
766
767 # The directory, where to install demo-files
768 # If we have some, we have to add following 2 lines to install rule:
769 #     $(MKDIR) $(DEMODIR); \
770 #     $(INSTALL) $(DEMO_FILES) $(DEMODIR); \
771 DEMODIR=$(DOCDIR)/demo
772
773 # We need this, because the documentation needs the classes and packages
774 # It's not really a good solution, but it's a working solution.
775 TEXINPUTS := $(PWD):$(TEXINPUTS)
776
777 #####
778 #   End of customization section
779 #####
780
781 LATEX = latex
782 PDFLATEX = pdflatex
783 TEX = TEX
784
785 COMMON_OPTIONS = # \OnlyDescription\CodelineNumbered\PageIndex
786 HISTORY_OPTIONS = \RecordChanges
787 DEVELOPER_OPTIONS = \EnableCrossrefs\RecordChanges\AlsoImplementation\CodelineIndex
788
789 # tarring options
790 EXgit = --exclude .git --exclude .gitignore --exclude auto --exclude tests \
791 --exclude *.tgz --exclude *.bib
792
793 # The name of the game
794 PACKAGE = struktex
795
796 DISTRIBUTION_FILES = ../$(PACKAGE)/$(PACKAGE).de.pdf \
797 ../$(PACKAGE)/$(PACKAGE).en.pdf \
798 ../$(PACKAGE)/$(PACKAGE).dtx \
799 ../$(PACKAGE)/$(PACKAGE).ins \
800 ../$(PACKAGE)/LIESMICH.md \
801 ../$(PACKAGE)/README.md
802 PACKAGE_FILES_A = $(subst ../$(PACKAGE)/,,$(DISTRIBUTION_FILES))
803 PACKAGE_FILES_B = $(subst $(PACKAGE).dtx ,,$(PACKAGE_FILES_A))
804 PACKAGE_FILES_C = $(subst $(PACKAGE).ins ,,$(PACKAGE_FILES_B))
805 PACKAGE_FILES_D = $(subst LIESMICH.md,,$(PACKAGE_FILES_C))
806 PACKAGE_FILES = $(subst README.md,,$(PACKAGE_FILES_D))
807
808 # To generate the version number of the distribution from the source
809 VERSION_L := git describe --long | xargs git --no-pager show -s \
810 --date=short --format=format:"$(PACKAGE) version ??? of %ad%n" |\
811 sed -e "s/???/'git describe --long'/"
812 VERSION_S := 'git describe --long | \

```

```

813             sed 's+-g.*+++'
814
815 # to create the correct tar-file
816 define TAR_COMMANDS
817 echo $$@
818 OUT_DIR=$(mktemp -d)
819 mkdir ${OUT_DIR}/struktex
820 cp $$@ ${OUT_DIR}/struktex
821 pushd ${OUT_DIR}
822 tar cvfz struktex.tgz struktex
823 popd
824 cp ${OUT_DIR}/struktex.tgz .
825 endef
826
827 export TAR_COMMANDS
828
829 ## Main Targets
830
831 # strip off the comments from the package
832 $(PACKAGE).sty $(PACKAGE)-test-*.tex: $(PACKAGE).ins $(PACKAGE).dtx $(PACKAGE).sed
833 +$(TEX) $<; \
834     source $(PACKAGE).makemake; \
835     make revision_no; \
836     source $(PACKAGE).sed # set version number
837
838 all-de: $(PACKAGE).de.pdf
839
840 all-en: $(PACKAGE).en.pdf
841
842 # prepare the file with git revision information
843 .PHONY: revision_no
844 revision_no: $(PACKAGE).sed
845
846 $(PACKAGE).sed: $(PACKAGE).dtx
847 printf "%b\n" "set_git_info() {" \
848     > $(PACKAGE).sed; \
849 printf "%b\n" "sed -i -e 's/^[\\t]*%% git revision information$/\\\" \
850     >> $(PACKAGE).sed; \
851 git describe --long | \
852     xargs git --no-pager show -s --format=format:\\
853     \" \\@git@ \\$Date: %ci $\\%\\%n\" >> $(PACKAGE).sed; \
854 git describe --long | cut -c 2- | \
855     sed -e "s/^/ \\$Revision: /" -e "s/$$/ $\\%\\%\\\" \
856     >> $(PACKAGE).sed; \
857 git describe --long | \
858     xargs git --no-pager show -s --format=format:" %%% \\$Author: %an $\\%\\%n" \
859     >> $(PACKAGE).sed; \
860 printf "%b\n" "/" \\$1" \
861     >> $(PACKAGE).sed; \
862 printf "%b\n" "};" \
863     >> $(PACKAGE).sed; \
864 printf "%b\n" "for f in \\\" \
865     >> $(PACKAGE).sed; \
866 printf "%b\n" "$(PACKAGE).sty \\\" \

```

```

867     >> $(PACKAGE).sed; \
868 printf "%b\n" "struktxf.sty \\\\" \
869     >> $(PACKAGE).sed; \
870 printf "%b\n" "struktxp.sty \\\\" \
871     >> $(PACKAGE).sed; \
872 printf "%b\n" "strukdoc.sty \\\\" \
873     >> $(PACKAGE).sed; \
874 printf "%b\n" "; do \\\\" \
875     >> $(PACKAGE).sed; \
876 printf "%b\n" " set_git_info \\\$f; done" \
877     >> $(PACKAGE).sed; \
878
879 # generate the documentation
880 $(PACKAGE).de.pdf: $(PACKAGE).dtx $(PACKAGE).sty struktex.sed
881 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{<}"
882 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{<}"
883 +mv $(<:.dtx=.pdf) $(<:.dtx=.de.pdf)
884
885 $(PACKAGE).en.pdf: $(PACKAGE).dtx $(PACKAGE).sty
886 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\primarylanguage{english}\input{<}"
887 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\primarylanguage{english}\input{<}"
888 +mv $(<:.dtx=.pdf) $(<:.dtx=.en.pdf)
889
890 # generate the documentation with revision history (only german)
891 history: $(PACKAGE).dtx $(PACKAGE).sty
892 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(HISTORY_OPTIONS)}\input{<}"
893 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(HISTORY_OPTIONS)}\input{<}"
894 +makeindex -s gind.ist $(PACKAGE).idx
895 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
896 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(HISTORY_OPTIONS)}\input{<}"
897
898 # generate the documentation for the developer (revision history always
899 # in german)
900 develop-de: $(PACKAGE).dtx $(PACKAGE).sty
901 +$(PDFLATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)}$(DEVELOPER_OPTIONS)}\input{<}"
902 +$(PDFLATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)}$(DEVELOPER_OPTIONS)}\input{<}"
903 +makeindex -s gind.ist $(PACKAGE).idx
904 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
905 +$(PDFLATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)}$(DEVELOPER_OPTIONS)}\input{<}"
906 +mv $(<:.dtx=.pdf) $(<:.dtx=.de.pdf)
907
908 develop-en: $(PACKAGE).dtx $(PACKAGE).sty
909 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(DEVELOPER_OPTIONS)}\def\primarylanguage{eng
910 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(DEVELOPER_OPTIONS)}\def\primarylanguage{eng
911 +makeindex -s gind.ist $(PACKAGE).idx
912 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
913 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(DEVELOPER_OPTIONS)}\def\primarylanguage{eng
914 +mv $(<:.dtx=.pdf) $(<:.dtx=.en.pdf)
915
916 # format the example/test files
917 test:
918 for i in `seq 1 3`; do \
919     f=$(PACKAGE)-test-$$i; \
920     echo file: $$f; \

```

```

921      $(PDFLATEX) $$f; \
922 done
923
924 install: $(PACKAGE).dtx $(PACKAGE).dvi
925 [ -d $(CLSDIR) ] || mkdir -p $(CLSDIR)
926 [ -d $(DOCDIR) ] || mkdir -p $(DOCDIR)
927 [ -d $(SRCDIR) ] || mkdir -p $(SRCDIR)
928 cp $(PACKAGE).sty      $(CLSDIR)
929 cp $(PACKAGE).dvi      $(DOCDIR)
930 cp $(PACKAGE).ins      $(SRCDIR)
931 cp $(PACKAGE).dtx      $(SRCDIR)
932 cp $(PACKAGE)-test-*.tex $(SRCDIR)
933 cp LIESMICH            $(SRCDIR)
934 cp README              $(SRCDIR)
935 cp THIS-IS-VERSION-$(VERSION) $(SRCDIR)
936
937 uninstall:
938 rm -f $(CLSDIR)/$(PACKAGE).sty
939 rm -fr $(DOCDIR)
940 rm -fr $(SRCDIR)
941
942 dist: $(PACKAGE).de.pdf $(PACKAGE).en.pdf $(PACKAGE).dtx $(PACKAGE).ins \
943 LIESMICH.md README.md
944 + echo "$$STAR_COMMANDS" > ./tar_commands; \
945 rm -f THIS_IS_VERSION_*; \
946 $(VERSION_L) | sed -e "s/???????/$(VERSION_S)/" > THIS_IS_VERSION_$(VERSION_S); \
947 sh ./tar_commands $^; \
948 mv ./struktex.tgz ./struktex-$(VERSION_S).tgz
949 rm ./tar_commands
950
951 tds-zip: $(PACKAGE_FILES)
952 + rm -f THIS_IS_VERSION_* *.zip; \
953 $(VERSION_L) | sed -e "s/???????/$(VERSION_S)/" > THIS_IS_VERSION_$(VERSION_S); \
954 DOC_FILES="LIESMICH.md README.md THIS_IS_* $(PACKAGE).???.pdf"; \
955 MAKE_FILES="$(PACKAGE).m*"; \
956 SRC_FILES="$(PACKAGE).dtx $(PACKAGE).ins"; \
957 STY_FILES="struk*.sty"; \
958 TEST_FILES="./$(PACKAGE)-test*"; \
959 SUPPORT_FILES="./$(PACKAGE).el"; \
960 if [[ -d /tmp/texmf ]]; then \
961   rm -rf /tmp/texmf; \
962 fi; \
963 if [[ -f $(PACKAGE)-TDS.zip ]]; then \
964   rm $(PACKAGE)-TDS.zip; \
965 fi; \
966 mkdir -p /tmp/texmf/doc/latex/$(PACKAGE); \
967 mkdir -p /tmp/texmf/source/latex/$(PACKAGE); \
968 mkdir -p /tmp/texmf/tex/latex/$(PACKAGE); \
969 cp -a $$DOC_FILES /tmp/texmf/doc/latex/$(PACKAGE); \
970 cp -a $$MAKE_FILES /tmp/texmf/doc/latex/$(PACKAGE); \
971 cp -a $$SRC_FILES /tmp/texmf/source/latex/$(PACKAGE); \
972 cp -a $$STY_FILES /tmp/texmf/tex/latex/$(PACKAGE); \
973 cp -a $$TEST_FILES /tmp/texmf/doc/latex/$(PACKAGE); \
974 cp -a $$SUPPORT_FILES /tmp/texmf/doc/latex/$(PACKAGE); \

```



```

975 VERSION_SHORT="xxx"; \
976 pushd /tmp/texmf; \
977 zip -r /tmp/$(PACKAGE)-TDS.zip .; \
978 popd; \
979 mv /tmp/$(PACKAGE)-TDS.zip ./$(PACKAGE)-TDS-$(VERSION_S).zip
980
981
982 clean:
983 -rm -f *.log *.aux *.brf *.idx *.ilg *.ind
984 -rm -f *.glg *.glo *.gls *.lof *.lot *.out *.toc *.tmp *~
985 -rm *.mk *.makemake
986
987 realclean: clean
988 -rm -f *.sty *.cls *.pdf
989 -rm -f *-test-* Makefile
990
991 clean-test:
992 rm $(PACKAGE)-test-*. * # this $-sign is needed for font-locking in XEmacs only
993 </makefile>

```

The following line – stripped off as `struktex.makemake` – can be used with the command

```
sh struktex.makemake
```

to generate the file `Makefile`, which can be further used to generate the documentation with a common `make` like the GNU `make`.

```

994 <*setup>
995 sed -e "‘echo \s/^ /@/g\” | tr ‘@’ ‘\011’“ struktex.mk > Makefile
996 </setup>

```

## 9 Style File for easier input while working with (X)emacs and AUCTEX

The (X)emacs and the package AUCTEX (<http://www.gnu.org/software/auctex/>) form a powerful tool for creating and editing of TEX/L<sup>A</sup>TEX files. If there is a suitable AUCTEX style file for a L<sup>A</sup>TEX package like the hereby provided StukTeX package, then there is support for many common operations like creating environments and so on. The following part provides such a style file; it must be copied to a place, where (X)emacs can find it after its creation.

This file is still in a development phase, i. e. one can work with it, but there is a couple of missing things as for example font locking or the automatic insertion of `\switch` commands according to the user's input.

```

997 <*auctex>
998 ;;; struktex.el --- AUCTEX style for ‘struktex.sty’
999
1000 ;; Copyright (C) 2006 - 2018 Free Software Foundation, Inc.
1001
1002 ;; Author: J. Hoffmann <j.hoffmann_(at)_fh-aachen.de>
1003 ;; Maintainer: j.hoffmann_(at)_fh-aachen.de
1004 ;; Created: 2017/06/06
1005 ;; Keywords: tex

```

```

1006
1007 ;;; Commentary:
1008 ;; This file adds support for 'struktex.sty'
1009
1010 ;;; Code:
1011
1012 (TeX-add-style-hook
1013 "struktex"
1014 (lambda ()
1015   ;; Add declaration to the list of environments which have
1016   ;; an optional argument for each item.
1017   (add-to-list 'LaTeX-item-list
1018     '("declaration" . LaTeX-item-argument))
1019   (LaTeX-add-environments
1020     "centernss"
1021     '("struktogramm" LaTeX-env-struktogramm)
1022     '("declaration" LaTeX-env-declaration))
1023   (TeX-add-symbols
1024     '("PositionNSS" 1)
1025     '("assert" [ "Height" ] "Assertion")
1026     '("assign" [ "Height" ] "Statement")
1027     "StrukTeX"
1028     '("case" TeX-mac-case)
1029     "switch" "Condition"
1030     "caseend"
1031     '("declarationtitle" "Title")
1032     '("description" "Name" "Meaning")
1033     "emptyset"
1034     '("exit" [ "Height" ] "What" )
1035     '("forever" TeX-mac-forever)
1036     "foreverend"
1037     '("forallin" TeX-mac-forallin)
1038     "forallin"
1039     '("ifthenelse" TeX-mac-ifthenelse)
1040     "change"
1041     "ifend"
1042     '("inparallel" TeX-mac-inparallel)
1043     '("task" "Description")
1044     "inparallelend"
1045     "sProofOn"
1046     "sProofOff"
1047     '("until" TeX-mac-until)
1048     "untilend"
1049     '("while" TeX-mac-while)
1050     "whileend"
1051     '("return" [ "Height" ] "Return value")
1052     '("sub" [ "Height" ] "Task")
1053     '("CenterNssFile" TeX-arg-file)
1054     '("centernssfile" TeX-arg-file))
1055   (TeX-run-style-hooks
1056     "pict2e"
1057     "emlines2"
1058     "curves"
1059     "struktxp"

```

```

1060     "struktxf"
1061     "ifthen")
1062     ;; Filling
1063     ;; Fontification
1064     ))
1065
1066 (defun LaTeX-env-struktogramm (environment)
1067   "Insert ENVIRONMENT with width, height specifications and
1068   optional title."
1069   (let ((width (read-string "Width: "))
1070         (height (read-string "Height: "))
1071         (title (read-string "Title (optional): ")))
1072     (LaTeX-insert-environment environment
1073                               (concat
1074                                (format "(%s,%s)" width height)
1075                                (if (not (zerop (length title)))
                                    (format "[%s]" title))))))
1076
1077 (defun LaTeX-env-declaration (environment)
1078   "Insert ENVIRONMENT with an optional title."
1079   (let ((title (read-string "Title (optional): ")))
1080     (LaTeX-insert-environment environment
1081                               (if (not (zerop (length title)))
                                    (format "[%s]" title))))))
1082
1083 (defun TeX-mac-case (macro)
1084   "Insert \\case with all arguments, the needed \\switch(es) and
1085   the final \\caseend. These are optional height and the required
1086   arguments slope, number of cases, condition, and the texts for
1087   the different cases"
1088   (let ((height (read-string "Height (optional): "))
1089         (slope (read-string "Slope: "))
1090         (number (read-string "Number of cases: "))
1091         (condition (read-string "Condition: "))
1092         (text (read-string "Case no. 1: "))
1093         (count 1)
1094         )
1095     (setq number-int (string-to-number number))
1096     (insert (concat (if (not (zerop (length height)))
                        (format "[%s]" height)
                        (format "{%s}{%s}{%s}{%s}"
                                slope number condition text))))
1097     (while (< count number-int)
1098       (end-of-line)
1099       (newline-and-indent)
1100       (newline-and-indent)
1101       (setq prompt (format "Case no. %d: " (+ 1 count)))
1102       (insert (format "\\switch{%s}" (read-string prompt)))
1103       (setq count (1+ count)))
1104     (end-of-line)
1105     (newline-and-indent)
1106     (newline-and-indent)
1107     (insert "\\caseend"))))
1108
1109
1110
1111
1112
1113

```

```

1114 (defun TeX-mac-forallin (macro)
1115   "Insert \\forallin-block with all arguments.
1116 This is the optional height and the description of the loop"
1117   (let ((height (read-string "Height (optional): "))
1118         (loop-description (read-string "Description of loop: ")))
1119     (insert (concat (if (not (zerop (length height)))
1120                        (format "[%s]" height))
1121                    (format "{%s}"
1122                        loop-description))))
1123   (end-of-line)
1124   (newline-and-indent)
1125   (newline-and-indent)
1126   (insert "\\forallinend"))
1127
1128 (defun TeX-mac-forever (macro)
1129   "Insert \\forever-block with all arguments.
1130 This is only the optional height"
1131   (let ((height (read-string "Height (optional): ")))
1132     (insert (if (not (zerop (length height)))
1133                (format "[%s]" height))
1134             (end-of-line)
1135             (newline-and-indent)
1136             (newline-and-indent)
1137             (insert "\\foreverend"))))
1138
1139 (defun TeX-mac-ifthenelse (macro)
1140   "Insert \\ifthenelse with all arguments.
1141 These are optional height and the required arguments
1142 left slope, right slope, condition, and the possible
1143 values of the condition"
1144   (let ((height (read-string "Height (optional): "))
1145         (lslope (read-string "Left slope: "))
1146         (rslope (read-string "Right slope: "))
1147         (condition (read-string "Condition: "))
1148         (conditionvl (read-string "Condition value left: "))
1149         (conditionvr (read-string "Condition value right: ")))
1150     (insert (concat (if (not (zerop (length height)))
1151                        (format "[%s]" height))
1152                    (format "{%s}{%s}{%s}{%s}{%s}"
1153                        lslope rslope condition conditionvl
1154                        conditionvr))))
1155   (end-of-line)
1156   (newline-and-indent)
1157   (newline-and-indent)
1158   (insert "\\change")
1159   (end-of-line)
1160   (newline-and-indent)
1161   (newline-and-indent)
1162   (insert "\\ifend"))
1163
1164 (defun TeX-mac-inparallel (macro)
1165   "Insert \\inparallel with all arguments, the needed \\task(s)
1166 and the final \\inparallelend. These are optional height and the
1167 required arguments number of tasks and the descriptions for the

```

```

1168 parallel tasks"
1169   (let ((height (read-string "Height (optional): "))
1170         (number (read-string "Number of parallel tasks: "))
1171         (text (read-string "Task no. 1: "))
1172         (count 1)
1173         )
1174     (setq number-int (string-to-number number))
1175     (insert (concat (if (not (zerop (length height)))
1176                       (format "[%s]" height))
1177                   (format "{%s}{%s}" number text)))
1178     (while (< count number-int)
1179       (end-of-line)
1180       (newline-and-indent)
1181       (newline-and-indent)
1182       (setq prompt (format "Task no. %d: " (+ 1 count)))
1183       (insert (format "\\task{%s}" (read-string prompt)))
1184       (setq count (1+ count)))
1185     (end-of-line)
1186     (newline-and-indent)
1187     (newline-and-indent)
1188     (insert "\\inparallelend"))
1189
1190 (defun TeX-mac-until (macro)
1191   "Insert \\until with all arguments.
1192 These are the optional height and the required argument condition"
1193   (let ((height (read-string "Height (optional): "))
1194         (condition (read-string "Condition: ")))
1195     (insert (concat (if (not (zerop (length height)))
1196                       (format "[%s]" height))
1197                   (format "{%s}" condition)))
1198     (end-of-line)
1199     (newline-and-indent)
1200     (newline-and-indent)
1201     (insert "\\untilend"))
1202
1203 (defun TeX-mac-while (macro)
1204   "Insert \\while with all arguments.
1205 These are the optional height and the required argument condition"
1206   (let ((height (read-string "Height (optional): "))
1207         (condition (read-string "Condition: ")))
1208     (insert (concat (if (not (zerop (length height)))
1209                       (format "[%s]" height))
1210                   (format "{-%s-}" condition)))
1211     (end-of-line)
1212     (newline-and-indent)
1213     (newline-and-indent)
1214     (insert "\\whileend"))
1215
1216 (defvar LaTeX-struktex-package-options '("anygradient" "curves" "debug"
1217                                           "draft" "emlines" "final"
1218                                           "fixedindent" "nofiller"
1219                                           "outer" "pict2e" "verification")
1220   "Package options for the struktex package.")
1221

```

1222 ;;; `struktex.el` ends here.  
1223 `</auctex>`

## References

- [Fut89] Gerald Futschek. *Programmentwicklung und Verifikation*. Springer Verlag, Wien – New York, 1989.
- [GMS94] Michel Goossens, Frank Mittelbach and Alexander Samarin. *The L<sup>A</sup>T<sub>E</sub>X-Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.
- [GMS04] Frank Mittelbach and Michel Goossens. *The L<sup>A</sup>T<sub>E</sub>X-Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, second edition, 2004.
- [Knu86] D. E. Knuth. *The T<sub>E</sub>X-Book*. Addison-Wesley, Reading, Massachusetts, 1986.
- [MDB94] Frank Mittelbach, Denys Duchier and Johannes Braams. *The DocStrip program*, Dezember 1994.
- [MDB01] Frank Mittelbach, Denys Duchier, Johannes Braams, Marcin Woliński and Mark Wooding *The DocStrip program*, September 2001.
- [Mit94] Frank Mittelbach. *The doc and shortvrb Packages*, Oktober 1994.
- [Mit01] Frank Mittelbach. *The doc and shortvrb Packages*, September 2001.
- [Neu96] Marion Neubauer. Feinheiten bei wissenschaftlichen Publikationen – Mikrotypographie-Regeln, Teil I. *Die T<sub>E</sub>Xnische Kom”odie*, 8(4):23–40, Februar 1996.
- [Rah92] Sebastian Rahtz. *The oz package*, 1992.