# struktex.sty*

Jobst Hoffmann
University of Applied Sciences Aachen, Abt. Jülich
Ginsterweg 1
52428 Jülich
Federal Republic of Germany

printed on July 9, 2017

**Abstract**

This article describes the use and implementation of LaTeX-*package* struktex.sty for structured box charts (Nassi-Shneiderman diagrams).

## Contents

## 1 License

This package is copyright © 1995 – 2017 by:

*This file has version number v2.2b-6-g08f43fa, last revised on 2017/07/09, documentation dated 2017/07/01.

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

## 2 Preface

StrukTEX 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

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

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 StrukTEX. It can generate the most important elements of a structured box chart like processing blocks, loops, mapping conventions for alternatives etc.[1]

Since version v-4.1a the mathematical symbols are loaded by AMS-TEX. 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 ($\varnothing$) 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 LATEX. – 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.

[1] Those who don't like to code the diagrams by hand, can use for example the program Struktorizer (`http://structorizer.fisch.lu/`). By using this program one can draw the diagrams by mouse and export the result into a LATEX-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 reimplementation of the `declaration`-environment through the `list`-environment by [GMS94, Abs. 3.3.4] (done in version v-4.5a).
3. The adaption to LaTeX $2_\varepsilon$ 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[2]

```
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 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 lieing before.

```
⟨*driver⟩
                                  % select the formatting language:
\expandafter\ifx\csname primarylanguage\endcsname\relax%
\def\primarylanguage{ngerman}%
\def\secondarylanguage{english}%
\else%
\def\secondarylanguage{ngerman}%
\fi

\documentclass[a4paper, \secondarylanguage,     % select the language
        \primarylanguage]{ltxdoc}

\PassOptionsToPackage{obeyspaces}{url} % must be done before any package is
                                  % loaded

\usepackage{babel}                % for switching the documentation language
\usepackage{strukdoc}             % the style-file for formatting this
                                  % documentation

```

[2]Generating the documentation is much easier with the `make` utility, see section 8.

```
\usepackage[pict2e, % <---------- to produce finer results
                                % visible under xdvi, alternatives are
                                % curves or emlines2 (visible only under
                                % ghostscript), leave out if not
                                % available
    verification,
    outer, % <------------------- to set the position of the \ifthenelse
                                % flags to the outer edges
    ]
    {struktex}
\GetFileInfo{struktex.sty}

\EnableCrossrefs
%\DisableCrossrefs   % say \DisableCrossrefs if index is ready

%\RecordChanges      % say \RecordChanges to gather update information

%\CodelineIndex      % say \CodelineIndex to index entry code by line number

\OnlyDescription    % say \OnlyDescription to omit the implementation details

\MakeShortVerb{\|}   % |\foo| acts like \verb+\foo+

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% to avoid underfull ... messages while formatting two/three columns
\hbadness=10000 \vbadness=10000

\typeout{\string\primarylanguage: \primarylanguage, \string\language: \the\language}

\def\languageNGerman{10}         % depends on language.dat, put
                                % \the\language here

\begin{document}
\makeatletter
\@ifundefined{selectlanguageEnglish}{}{\selectlanguage{english}}
\makeatother
\DocInput{struktex.dtx}
\end{document}
⟨/driver⟩
```

## 4 The User interface

The struktex.sty will be included in a 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 ∅ in alternatives.

5. `draft`, `final`:

   These options serve as usual to diffentiate 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

`\nat` `\integer` `\real` `\complex` `\emptyset` `\MathItalics` `\MathNormal`

Since sets of natural, whole, real and complex numbers ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$ and $\mathbb{C}$) occur often in the Mathematics Mode they can be reached by the macros `\nat`, `\integer`, `\real` and `\complex`. Similarly "'∅"', which is generated by `\emptyset`, is the more remarkable symbol for the empty statement than the standard symbol "'$\emptyset$"'. Other set symbols like $\mathbb{L}$ (for solution space) have to be generated by `$\mathbb L$`.

One can influence the descriptions of variable names by these macros.

$$NewValue = OldValue + Correction$$

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

und

$$NewValue = OldValue + Correction$$

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

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

\pVariable
\pVar
\pKeyword
\pKey
\pComment

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 struktxp.sty to be able to use them in another context. From version 122 on struktxp.sty 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

```
\PassOptionsToPackage{obeyspaces}{url}
```

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

```
\usepackage{struktex}
```

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:

```
cANormalVariable
c_a_normal_variable
&iAddressOfAVariable
pPointerToAVariable^.sContent
```

```
\obeylines
 \pVariable{cANormalVariable}
 \pVariable{c_a_normal_variable}
 \pVariable{&iAddressOfAVariable}
 \pVariable{pPointerToAVariable^.sContent}
```

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:

```
begin
program
#include
```

```
\obeylines
 \pKeyword{begin}
 \renewcommand{\pLanguage}{Pascal}
 \pKeyword{program}
 \renewcommand{\pLanguage}{C}
 \pKeyword{#include}
```

`\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` `\pFalse` `\pFonts` `\pBoolValue`

Boolean values play an importand role in programming. There are given adequate values by `\pTrue` and `\pFalse`: `true` and `false`.

The macro `\pFonts` is used for the choice of fonts for representation of variables, keywords and comments:

`\pFonts{`⟨*variablefont*⟩`}{`⟨*keywordfont*⟩`} {`⟨*commentfont*⟩`}`

The default values for the certain fonts are

- ⟨*variablefont*⟩ as `\small\sffamily`,
- ⟨*keywordfont*⟩ as `\small\sffamily\bfseries` and
- ⟨*commentfont*⟩ 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{`⟨*Yes-Value*⟩`}{`⟨*No-Value*⟩`}`

So the lines

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

result in the following:

*no*= `not` *yes*

`\sVar` `\sKey` `\sTrue` `\sFalse`

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

## 4.3 The Macros for generating structured box charts

`struktogramm` `\sProofOn` `\sProofOff` `\PositionNSS`

The environment

```
\begin{struktogramm}(⟨width⟩,⟨height⟩)[⟨titel⟩]
...
\end{struktogramm}
```

generates space for a new box chart. Both the parameters provide the width and the height of the place, which is reservated for the structured box chart. Lengths etc. are described in millimeters. In doing so the actual value of `\unitlength` is unimportand. 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 does'nt 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 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 StrukTeX `\unitlength` is of the same quantity as before. But it is redefined within a structured box chart and need not be changed there.

`\assign`

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

```
\assign[⟨height⟩]{⟨content⟩},
```

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.

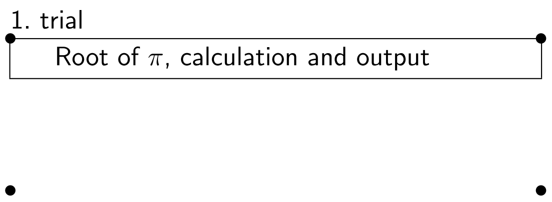
**Example 1**

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.



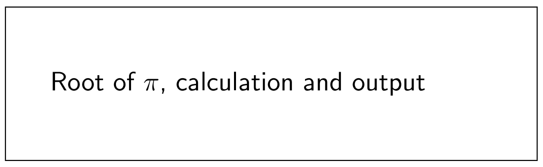
The meaning of the optional argument will be made clear by the following example:

**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.



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

`\begin{declaration}[`⟨*titel*⟩`]`

. . .

`\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{`⟨*title*⟩`}`. If one wants to generate a special title for a certain structured box chart, one has to declare it within square brackets.

`\description`
`\descriptionindent`
`\descriptionwidth`
`\descriptionsep`

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

`\description{`⟨*variableName*⟩`}{`⟨*variableDescription*⟩`}`

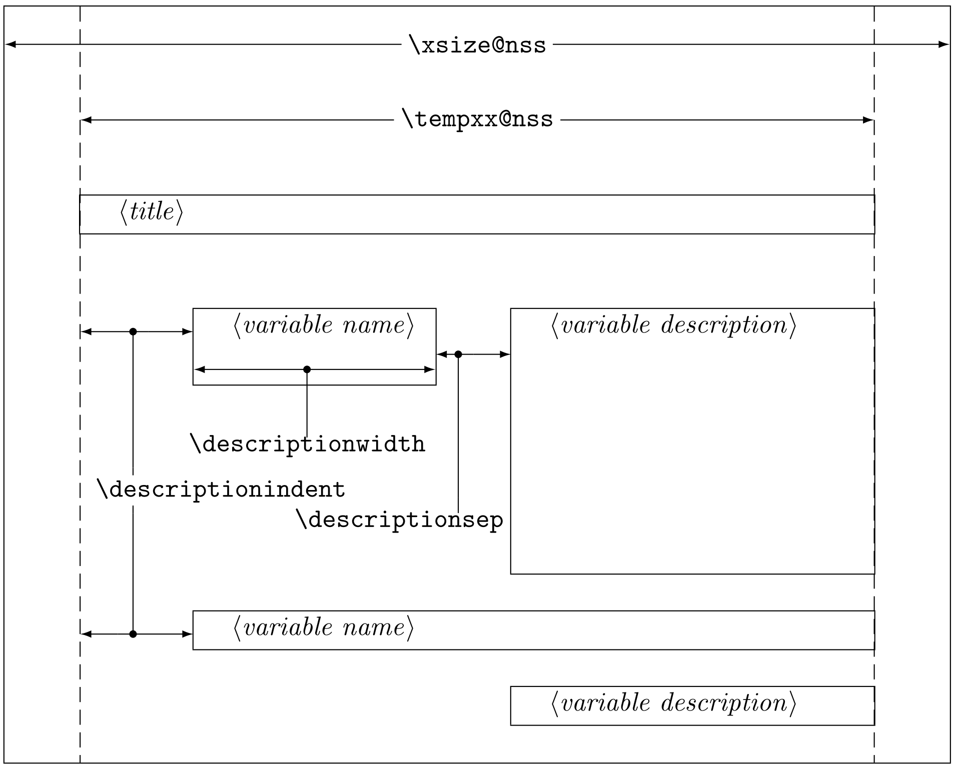


Figure 1: Construction of a Variable Description

In doing so one has to pay attention on the ⟨*variableName*⟩, 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 StrukTEX). 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 titels generated by the empty square brackets.

```
 providing memory space:
     iVar      {an  int  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:

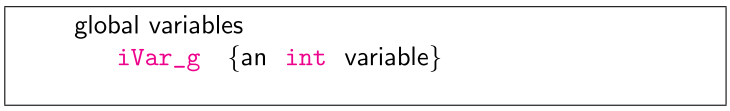
```
 Parameter:
     iPar      {an  int  parameter with the meaning
               described here}
local Variables:
     iVar      {an  int  variable with the meaning de-
               scribed here}
     dVar      {a  double  variable with the meaning
               described here}
```

Finally the global declaration of a titel:

```
\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 TEX.

`\sub`
`\return`

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

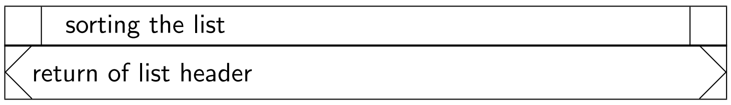
`\sub[`⟨*height*⟩`]{`⟨*text*⟩`}`

`\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`
`\whileend`
`\until`
`\untilend`
`\forallin`
`\forallinend`
`\forever`
`\foreverend`

For representation of loop constructions there are three instructions available: `\while`, `\until` and `\forever`. The while loop is a repetition with preceding condition check (loop with a pre test). The until loop checks the condition at the end of the loop (loop with a post test). And the forever loop is a neverending loop, that can be left by `\exit`.

`\while[`⟨*width*⟩`]{`⟨*text*⟩`}`⟨*structured subbox chart*⟩

`\whileend`

`\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 is'nt 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 StrukTeX (except `\openstrukt` and `\closestrukt`), which build up the box chart within the `\while` loop, the `\until` loop or the `\forever` loop.

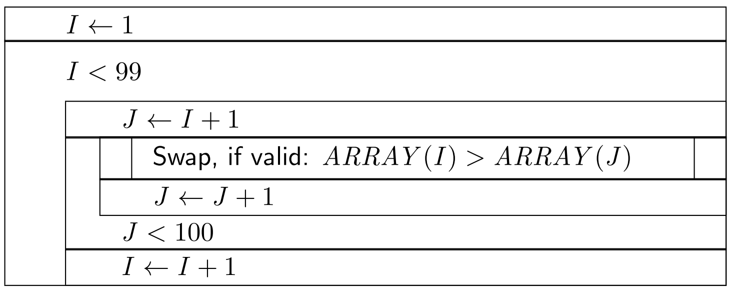
For compatibility with further development of the struktex.sty of J. Dietel there are the macros `\dfr` and `\dfrend` with the same meaning as `\forever` and `\foreverend`.

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}
```

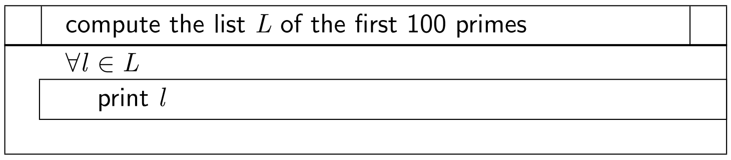
These instructions lead to the following structured box chart:



**Example 6**

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

For the representation of alternatives StrukTeX provides mapping conventions for an If-Then-Else-block and a Case-construction for multiple alternatives. Since in the traditional `picture` environment of 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[`⟨*height*⟩`]{`⟨*left angle*⟩`}{`⟨*right angle*⟩`}`

    `{`⟨*condition*⟩`}{`⟨*left text*⟩`}{`⟨*right text*⟩`}`

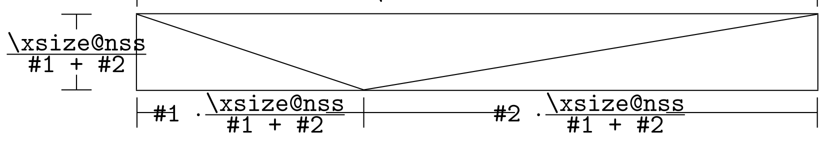
  ⟨*structured subbox chart*⟩

`\change`

  ⟨*structured subbox chart*⟩

`\ifend`

In the case of omitting the optional argument ⟨*height*⟩ ⟨*left angle*⟩ and ⟨*right angle*⟩ 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 ⟨*height*⟩ is given, then this value determines the height of the conditioning rectangle instead of the expression $\frac{\texttt{\textbackslash xsize@nss}}{\texttt{\#1 + \#2}}$.

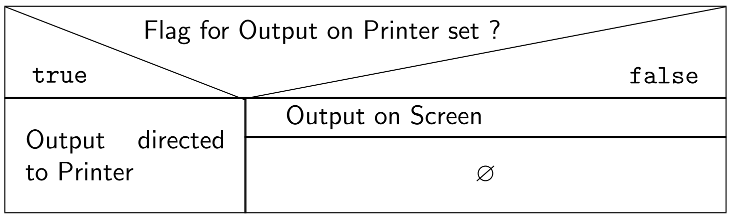


⟨*condition*⟩ is set in the upper triangle built in the above way. The parameters ⟨*left text*⟩ and ⟨*right text*⟩ are set in the left or right lower triangle respectively. The conditioning text can be made up in its triangle box. From version v-5.3 on the conditioning text …[3] Both the other texts 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 ∅ 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:

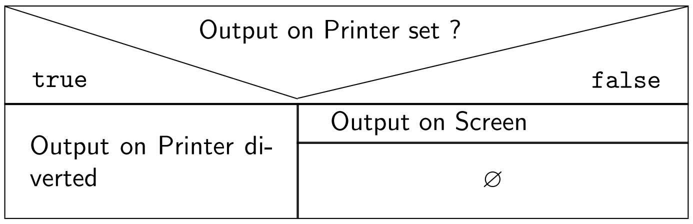


**Example 8**

```
\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:

[3]This extension is due to Daniel Hagedorn, whom I have to thank for his work.

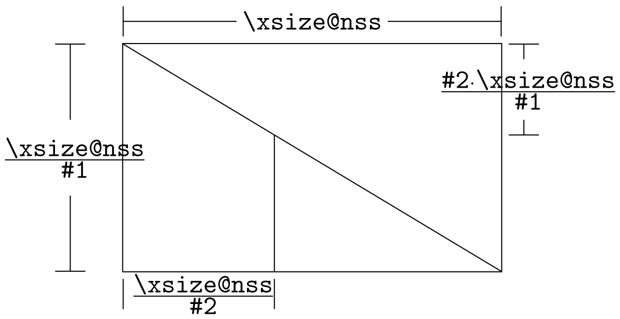


`\case` `\switch` `\caseend`

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:


\xsize@nss
$\frac{\#2\cdot\backslash xsize@nss}{\#1}$
$\frac{\backslash xsize@nss}{\#1}$
$\frac{\backslash xsize@nss}{\#2}$

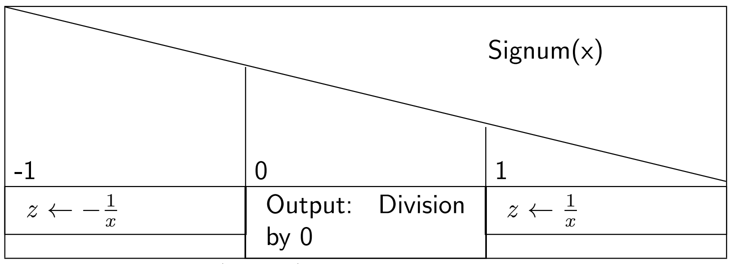
The second parameter ⟨*number of cases*⟩ specifies the number of cases, that have to be drawn. All structured subbox charts of the certain cases get the same width. The ⟨*text of 1. case*⟩ has to be given as a parameter of the `\case` instruction. All other cases are introduced by the `\switch` instruction. 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**

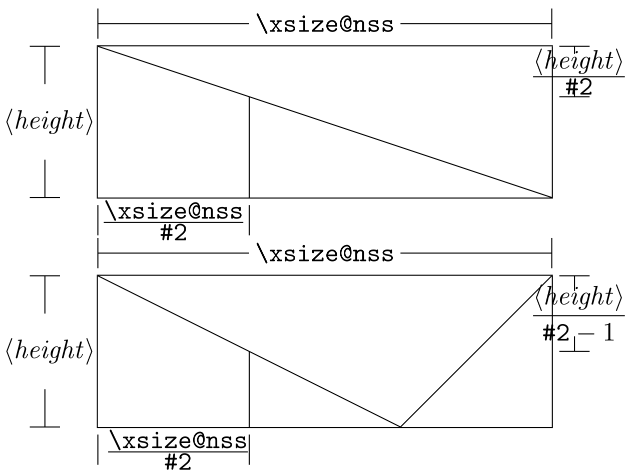
```
\begin{struktogramm}(95,30)
    \case{4}{3}{Signum(x)}{-1}
        \assign{$z \gets - \frac{1}{x}$}
    \switch{0}
```

```
        \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 scrumbled 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.

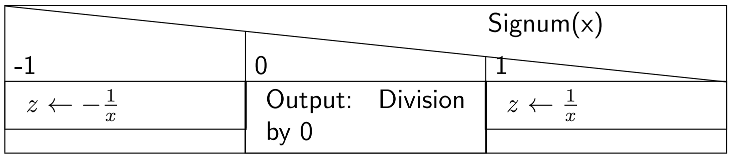

\xsize@nss
⟨height⟩
$\frac{\langle height\rangle}{\#2}$
$\frac{\texttt{\textbackslash xsize@nss}}{\#2}$
\xsize@nss
⟨height⟩
$\frac{\langle height\rangle}{\#2-1}$
$\frac{\texttt{\textbackslash xsize@nss}}{\#2}$

**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}
```

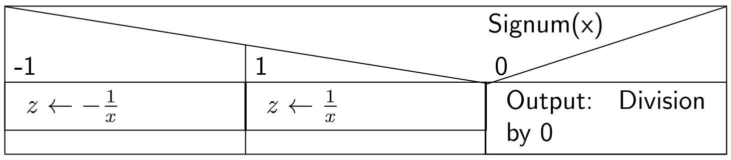
These instructions lead to the following structured box chart:



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:

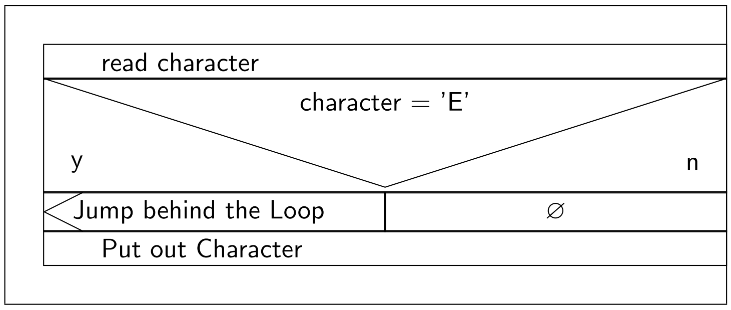


The following example shows, how one can exit a neverending loop by a simple branch. The example is transferable to a multiple branch without much effort.
**Example 12**

```
\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}
```

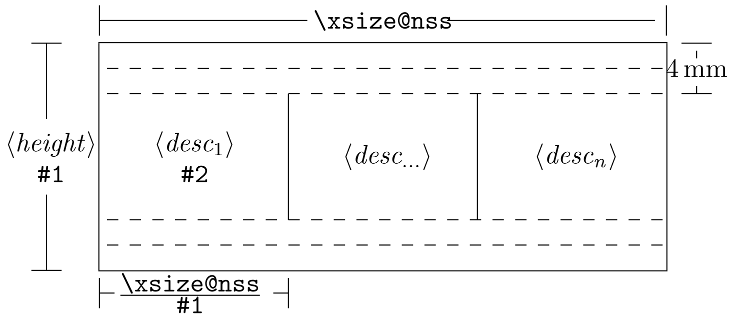
These instructions lead to the following structured box chart:



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

`\inparallel[`⟨*height of 1*$^{st}$ *task*⟩`]{`⟨*number of parallel tasks*⟩`}{`⟨*description of 1*$^{st}$ *task*⟩`}}`
`\task[`⟨*position*⟩`]{`⟨*description of 2*$^{nd}$ *task*⟩`}`
...
`\task[`⟨*position*⟩`]{`⟨*description of n*$^{th}$ *task*⟩`}`
`\inparallelend`

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

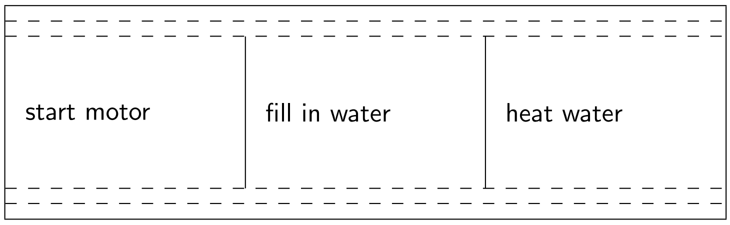


Note: the tasks are not allowed to get divided by `\assign` or so. If one needs some finer description of a task, this should be made outside of the current structured box chart.
**Example 13** (Application of `\inparallel`)

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

```
        \task{fill in water}
        \task{heat water}
        \inparallelend
    \end{struktogramm}
```

These instructions produce the following structured box chart:



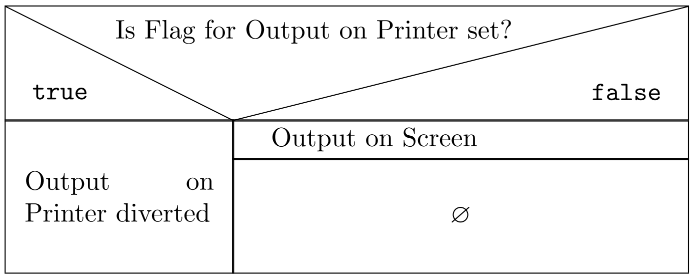
`centernss` If a structured box chart shall be represented centered, then the environment

```
\begin{centernss}
  ⟨Struktogramm⟩
\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:


Is Flag for Output on Printer set?
true
false
Output on Printer diverted
Output on Screen
∅

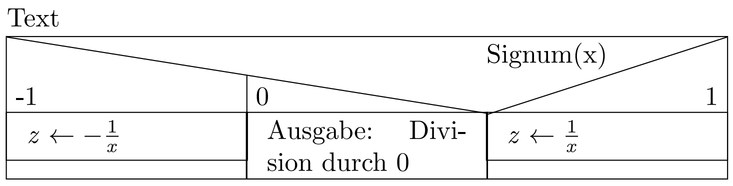
`\CenterNssFile` In many cases structured box charts are recorded in particular files such, that they can be tested seperately, if they are correct, or that they can be used in other connections. If they should be included centeredly, then one can not use the following construction:

```
\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:



`\openstrukt` `\closestrukt` These two macros are only preserved because of compatibility reasons with previous versions of StrukTeX. 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`:

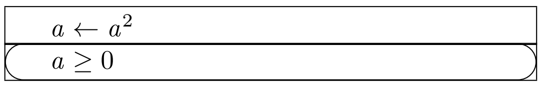
`\assert[`⟨*height*⟩`]{`⟨*assertion*⟩`}`,

It's usage can be seen from the following:

```
\begin{struktogramm}(70,20)[Assertions in structured box charts]
    \assign{\(a\gets a^2\)}
    \assert{\(a\ge0\)}
\end{struktogramm}
\sProofOff
```

The resulting structured box chart looks like

Assertions in structured box charts



## 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.

```
⟨*example1⟩
\begin{struktogramm}(95,40)[Text]
    \case[10]{3}{3}{Signum(x)}{-1}
        \assign{\(z \gets - \frac{1}{x}\)}
    \switch{0}
        \assign{Ausgabe: Division durch 0}
    \switch[r]{1}
        \assign{\(z \gets \frac{1}{x}\)} \caseend
\end{struktogramm}
⟨/example1⟩
```

## 6 Some example files

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

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

```
⟨*example2⟩
\documentclass[draft]{article}
\usepackage{struktex}

\begin{document}

\begin{struktogramm}(90,137)
    \assign%
    {
      \begin{declaration}[]
          \description{\(a, b, c\)}{three variables which are to be sorted}
          \description{\(tmp\)}{temporary variable for the circular swap}
      \end{declaration}
    }
    \ifthenelse{1}{2}{\(a\le c\)}{j}{n}
    \change
    \assign{\(tmp\gets a\)}
    \assign{\(a\gets c\)}
    \assign{\(c\gets tmp\)}
    \ifend
    \ifthenelse{2}{1}{\(a\le b\)}{j}{n}
    \ifthenelse{1}{1}{\(b\le c\)}{j}{n}
    \change
    \assign{\(tmp\gets c\)}
    \assign{\(c\gets b\)}
    \assign{\(b\gets tmp\)}
    \ifend
    \change
    \assign{\(tmp\gets a\)}
    \assign{\(a\gets b\)}
    \assign{\(b\gets tmp\)}
    \ifend
\end{struktogramm}

\end{document}
⟨/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.

```
⟨*example3⟩
\documentclass{article}
\usepackage[pict2e, verification]{struktex}

\begin{document}
\def\StruktBoxHeight{7}
%\sProofOn{}
\begin{struktogramm}(90,137)
    \assign%
    {
      \begin{declaration}[]
          \description{\(a, b, c\)}{three variables which are to be sorted}
          \description{\(tmp\)}{temporary variable for the circular swap}
      \end{declaration}
    }
    \assert[\StruktBoxHeight]{\sTrue}
    \ifthenelse[\StruktBoxHeight]{1}{2}{\(a\le c\)}{j}{n}
        \assert[\StruktBoxHeight]{\(a\le c\)}
    \change
        \assert[\StruktBoxHeight]{\(a>c\)}
        \assign[\StruktBoxHeight]{\(tmp\gets a\)}
        \assign[\StruktBoxHeight]{\(a\gets c\)}
        \assign[\StruktBoxHeight]{\(c\gets tmp\)}
        \assert[\StruktBoxHeight]{\(a<c\)}
    \ifend
    \assert[\StruktBoxHeight]{\(a\le c\)}
    \ifthenelse[\StruktBoxHeight]{2}{1}{\(a\le b\)}{j}{n}
        \assert[\StruktBoxHeight]{\(a\le b \wedge a\le c\)}
        \ifthenelse[\StruktBoxHeight]{1}{1}{\(b\le c\)}{j}{n}
            \assert[\StruktBoxHeight]{\(a\le b \le c\)}
        \change
            \assert[\StruktBoxHeight]{\(a \le  c<b\)}
            \assign[\StruktBoxHeight]{\(tmp\gets c\)}
            \assign[\StruktBoxHeight]{\(c\gets b\)}
            \assign[\StruktBoxHeight]{\(b\gets tmp\)}
            \assert[\StruktBoxHeight]{\(a\le b<c\)}
        \ifend
    \change
        \assert[\StruktBoxHeight]{\(b < a\le c\)}
        \assign[\StruktBoxHeight]{\(tmp\gets a\)}
        \assign[\StruktBoxHeight]{\(a\gets b\)}
        \assign[\StruktBoxHeight]{\(b\gets tmp\)}
        \assert[\StruktBoxHeight]{\(a<b\le c\)}
    \ifend
    \assert[\StruktBoxHeight]{\(a\le b \le c\)}
\end{struktogramm}

\end{document}
⟨/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}`.

```
⟨*example4⟩
\documentclass[english]{article}

\usepackage{babel}
\usepackage{struktex}

\nofiles

\begin{document}

\pLanguage{Pascal}
\section*{Default values (Pascal):}

{\obeylines
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{After changing the boolean values with}
\verb-\pBoolValue{yes}{no}-:

{\obeylines
\pBoolValue{yes}{no}
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:

{\obeylines
\pFonts{\itshape}{\sffamily\bfseries}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:

{\obeylines
\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\bfseries\itshape}{}-:

{\obeylines
\pFonts{\itshape}{\bfseries\itshape}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse

\vspace{15pt}
Without \textit{italic correction}:
    M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
}

\pLanguage{C}
\pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
\section*{Default values (C):}

{\obeylines
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{After changing the boolean values with}
\verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:

{\obeylines
\pBoolValue{\texttt{yes}}{\texttt{no}}
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:
```

```
{\obeylines
\pFonts{\itshape}{\sffamily\bfseries}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:

{\obeylines
\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\bfseries\itshape}{}-:

{\obeylines
\pFonts{\itshape}{\bfseries\itshape}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse

\vspace{15pt}
Without \textit{italic correction}:
    M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
}

\pLanguage{Java}
\pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
\section*{Default values (Java):}

{\obeylines
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{After changing the boolean values with}
\verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:

{\obeylines
\pBoolValue{\texttt{yes}}{\texttt{no}}
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

```

```
\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:

{\obeylines
\pFonts{\itshape}{\sffamily\bfseries}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:

{\obeylines
\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\bfseries\itshape}{}-:

{\obeylines
\pFonts{\itshape}{\bfseries\itshape}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse

\vspace{15pt}
Without \textit{italic correction}:
    M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
}

\pLanguage{Python}
\pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
\section*{Default values (Python):}

{\obeylines
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{After changing the boolean values with}
\verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:

{\obeylines
\pBoolValue{\texttt{yes}}{\texttt{no}}
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:

{\obeylines
\pFonts{\itshape}{\sffamily\bfseries}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:

{\obeylines
\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

\paragraph{after changing the fonts with}
\verb-\pFonts{\itshape}{\bfseries\itshape}{}-:

{\obeylines
\pFonts{\itshape}{\bfseries\itshape}{}
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse

\vspace{15pt}
Without \textit{italic correction}:
    M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
}

\end{document}
%%
%% End of file ‘struktex-test-2.tex’.
⟨/example4⟩
```

## 6.4 Example file for testing the macros of **struktxp.sty**

```
⟨*example5⟩
\documentclass{article}

\usepackage{struktxp,struktxf}

\makeatletter
\newlength{\fdesc@len}
\newcommand{\fdesc@label}[1]%
```

```
{%
    \settowidth{\fdesc@len}{{\fdesc@font #1}}%
    \advance\hsize by -2em
    \ifdim\fdesc@len>\hsize%               % term > labelwidth
        \parbox[b]{\hsize}%
        {%
            \fdesc@font #1%
        }\\%
    \else%                                      % term < labelwidth
    \ifdim\fdesc@len>\labelwidth%               % term > labelwidth
        \parbox[b]{\labelwidth}%
        {%
            \makebox[0pt][l]{{\fdesc@font #1}}\\%
        }%
    \else%                                      % term < labelwidth
        {\fdesc@font #1}%
    \fi\fi%
    \hfil\relax%
}
\newenvironment{fdescription}[1][\tt]%
{%
    \def\fdesc@font{#1}
    \begin{quote}%
    \begin{list}{}%
    {%
        \renewcommand{\makelabel}{\fdesc@label}%
        \setlength{\labelwidth}{120pt}%
        \setlength{\leftmargin}{\labelwidth}%
        \addtolength{\leftmargin}{\labelsep}%
    }%
}%
{%
    \end{list}%
    \end{quote}%
}
\makeatother

\pLanguage{Java}

\begin{document}

\begin{fdescription}
 \item[\index{Methoden>drawImage(Image img,
                                   int dx1,
                                   int dy1,
                                   int dx2,
                                   int dy2,
                                   int sx1,
                                   int sy1,
                                   int sx2,
                                   int sy2,
                                   ImageObserver observer)=%
       \Expr{\pKey{public} \pKey{abstract} \pKey{boolean} drawImage(Image img,
                                   \pKey{int} dx1,
                                    \pKey{int} dy1,
                                    \pKey{int} dx2,
                                    \pKey{int} dy2,
                                    \pKey{int} sx1,
                                    \pKey{int} sy1,
                                    \pKey{int} sx2,
                                    \pKey{int} sy2,
                                    ImageObserver observer)}}%
       \pExp{public abstract boolean drawImage(Image img, int dx1, int
         dy1, int dx1, int dy2, int sx1, int sy1, int sx2, int sy2,
         ImageObserver observer)}]%
 \ldots
\end{fdescription}
\end{document}
%%
%% End of file ‘struktex-test-5.tex’.
⟨/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".

```
⟨*strukdoc⟩
\RequirePackage{ifpdf}
\newif\ifcolor \IfFileExists{color.sty}{\colortrue}{}
\ifpdf \RequirePackage[colorlinks]{hyperref}\else
    \def\href#1{\texttt}\fi
\ifcolor \RequirePackage{color}\fi
\RequirePackage{nameref}
\RequirePackage{url}
\renewcommand\ref{\protect\T@ref}
\renewcommand\pageref{\protect\T@pageref}
\@ifundefined{zB}{}{\endinput}
\providecommand\pparg[2]{%
  {\ttfamily(}\meta{#1},\meta{#2}{\ttfamily)}}
\providecommand\envb[1]{%
  {\ttfamily\char‘\\begin\char‘\{#1\char‘\}}}
\providecommand\enve[1]{%
  {\ttfamily\char‘\\end\char‘\{#1\char‘\}}}
\newcommand{\zBspace}{z.\,B.}
\let\zB=\zBspace
\newcommand{\dhspace}{d.\,h.}
\let\dh=\dhspace
\let\foreign=\textit
\newcommand\Abb[1]{Abbildung~\ref{#1}}
\def\newexample#1{%
```

```
  \@ifnextchar[{\@oexmpl{#1}}{\@nexmpl{#1}}}
\def\@nexmpl#1#2{%
  \@ifnextchar[{\@xnexmpl{#1}{#2}}{\@ynexmpl{#1}{#2}}}
\def\@xnexmpl#1#2[#3]{%
  \expandafter\@ifdefinable\csname #1\endcsname
    {\@definecounter{#1}\@newctr{#1}[#3]%
     \expandafter\xdef\csname the#1\endcsname{%
       \expandafter\noexpand\csname the#3\endcsname \@exmplcountersep
          \@exmplcounter{#1}}%
     \global\@namedef{#1}{\@exmpl{#1}{#2}}%
     \global\@namedef{end#1}{\@endexample}}}
\def\@ynexmpl#1#2{%
  \expandafter\@ifdefinable\csname #1\endcsname
    {\@definecounter{#1}%
     \expandafter\xdef\csname the#1\endcsname{\@exmplcounter{#1}}%
     \global\@namedef{#1}{\@exmpl{#1}{#2}}%
     \global\@namedef{end#1}{\@endexample}}}
\def\@oexmpl#1[#2]#3{%
  \@ifundefined{c@#2}{\@nocounterr{#2}}%
    {\expandafter\@ifdefinable\csname #1\endcsname
    {\global\@namedef{the#1}{\@nameuse{the#2}}%
  \global\@namedef{#1}{\@exmpl{#2}{#3}}%
  \global\@namedef{end#1}{\@endexample}}}}
\def\@exmpl#1#2{%
  \refstepcounter{#1}%
  \@ifnextchar[{\@yexmpl{#1}{#2}}{\@xexmpl{#1}{#2}}}
\def\@xexmpl#1#2{%
  \@beginexample{#2}{\csname the#1\endcsname}\ignorespaces}
\def\@yexmpl#1#2[#3]{%
  \@opargbeginexample{#2}{\csname the#1\endcsname}{#3}\ignorespaces}
\def\@exmplcounter#1{\noexpand\arabic{#1}}
\def\@exmplcountersep{.}
\def\@beginexample#1#2{%
    \@nobreaktrue\list{}{\setlength{\rightmargin}{\leftmargin}}%
    \item[{\bfseries #1\ #2}]\mbox{}\\\sf}
\def\@opargbeginexample#1#2#3{%
    \@nobreaktrue\list{}{\setlength{\rightmargin}{\leftmargin}}%
    \item[{\bfseries #1\ #2}\ (#3)]\mbox{}\\\sf}
\def\@endexample{\endlist}

\newexample{tExample}{\ifnum\language=\languageNGerman Beispiel\else Example\fi}

\newwrite\struktex@out
\newenvironment{example}%
 {\begingroup% Lets keep the changes local
  \@bsphack
  \immediate\openout \struktex@out \jobname.tmp
  \let\do\@makeother\dospecials\catcode`\^^M\active
  \def\verbatim@processline{%
    \immediate\write\struktex@out{\the\verbatim@line}}%
  \verbatim@start}%
 {\immediate\closeout\struktex@out\@esphack\endgroup%
%
% And here comes the part of Tobias Oetiker
```

```
%
  \par\small\addvspace{3ex plus 1ex}\vskip -\parskip
  \noindent
  \makebox[0.45\linewidth][l]{%
  \begin{minipage}[t]{0.45\linewidth}
    \vspace*{-2ex}
    \setlength{\parindent}{0pt}
    \setlength{\parskip}{1ex plus 0.4ex minus 0.2ex}
    \begin{trivlist}
      \item\input{\jobname.tmp}
    \end{trivlist}
  \end{minipage}}%
  \hfill%
  \makebox[0.5\linewidth][l]{%
  \begin{minipage}[t]{0.50\linewidth}
    \vspace*{-1ex}
    \verbatiminput{\jobname.tmp}
  \end{minipage}}
  \par\addvspace{3ex plus 1ex}\vskip -\parskip
}

\newtoks\verbatim@line
\def\verbatim@startline{\verbatim@line{}}
\def\verbatim@addtoline#1{%
  \verbatim@line\expandafter{\the\verbatim@line#1}}
\def\verbatim@processline{\the\verbatim@line\par}
\def\verbatim@finish{\ifcat$\the\verbatim@line$\else
  \verbatim@processline\fi}

\def\verbatimwrite#1{%
  \@bsphack
  \immediate\openout \struktex@out #1
  \let\do\@makeother\dospecials
  \catcode`\^^M\active \catcode`\^^I=12
  \def\verbatim@processline{%
    \immediate\write\struktex@out
      {\the\verbatim@line}}%
  \verbatim@start}
\def\endverbatimwrite{%
  \immediate\closeout\struktex@out
  \@esphack}

\@ifundefined{vrb@catcodes}%
  {\def\vrb@catcodes{%
     \catcode`\!12\catcode`\[12\catcode`\]12}}{}
\begingroup
 \vrb@catcodes
 \lccode`\!=`\\ \lccode`\[=`\{ \lccode`\]=`\}
 \catcode`\~=\active \lccode`\~=`\^^M
 \lccode`\C=`\C
 \lowercase{\endgroup
    \def\verbatim@start#1{%
      \verbatim@startline
      \if\noexpand#1\noexpand~%
      \let\next\verbatim@
    \else \def\next{\verbatim@#1}\fi
    \next}%
  \def\verbatim@#1~{\verbatim@@#1!end\@nil}%
  \def\verbatim@@#1!end{%
     \verbatim@addtoline{#1}%
     \futurelet\next\verbatim@@@}%
  \def\verbatim@@@#1\@nil{%
     \ifx\next\@nil
       \verbatim@processline
       \verbatim@startline
       \let\next\verbatim@
     \else
       \def\@tempa##1!end\@nil{##1}%
       \@temptokena{!end}%
       \def\next{\expandafter\verbatim@test\@tempa#1\@nil~}%
     \fi \next}%
  \def\verbatim@test#1{%
         \let\next\verbatim@test
         \if\noexpand#1\noexpand~%
           \expandafter\verbatim@addtoline
             \expandafter{\the\@temptokena}%
           \verbatim@processline
           \verbatim@startline
           \let\next\verbatim@
         \else \if\noexpand#1
           \@temptokena\expandafter{\the\@temptokena#1}%
         \else \if\noexpand#1\noexpand[%
           \let\@tempc\@empty
           \let\next\verbatim@testend
         \else
           \expandafter\verbatim@addtoline
             \expandafter{\the\@temptokena}%
           \def\next{\verbatim@#1}%
         \fi\fi\fi
         \next}%
  \def\verbatim@testend#1{%
       \if\noexpand#1\noexpand~%
         \expandafter\verbatim@addtoline
           \expandafter{\the\@temptokena[}%
         \expandafter\verbatim@addtoline
           \expandafter{\@tempc}%
         \verbatim@processline
         \verbatim@startline
         \let\next\verbatim@
       \else\if\noexpand#1\noexpand]%
         \let\next\verbatim@@testend
       \else\if\noexpand#1\noexpand!%
         \expandafter\verbatim@addtoline
           \expandafter{\the\@temptokena[}%
         \expandafter\verbatim@addtoline
           \expandafter{\@tempc}%
         \def\next{\verbatim@!}%
       \else \expandafter\def\expandafter\@tempc\expandafter
```

```
          {\@tempc#1}\fi\fi\fi
        \next}%
    \def\verbatim@@testend{%
       \ifx\@tempc\@currenvir
         \verbatim@finish
         \edef\next{\noexpand\end{\@currenvir}%
                    \noexpand\verbatim@rescan{\@currenvir}}%
       \else
         \expandafter\verbatim@addtoline
           \expandafter{\the\@temptokena[}%
           \expandafter\verbatim@addtoline
             \expandafter{\@tempc]}%
         \let\next\verbatim@
       \fi
       \next}%
    \def\verbatim@rescan#1#2~{\if\noexpand~\noexpand#2~\else
        \@warning{Characters dropped after `\string\end{#1}'}\fi}}

\newread\verbatim@in@stream
\def\verbatim@readfile#1{%
  \verbatim@startline
  \openin\verbatim@in@stream #1\relax
  \ifeof\verbatim@in@stream
    \typeout{No file #1.}%
  \else
    \@addtofilelist{#1}%
    \ProvidesFile{#1}[(verbatim)]%
    \expandafter\endlinechar\expandafter\m@ne
    \expandafter\verbatim@read@file
    \expandafter\endlinechar\the\endlinechar\relax
    \closein\verbatim@in@stream
  \fi
  \verbatim@finish
}
\def\verbatim@read@file{%
  \read\verbatim@in@stream to\next
  \ifeof\verbatim@in@stream
  \else
    \expandafter\verbatim@addtoline\expandafter{\expandafter\check@percent\next}%
    \verbatim@processline
    \verbatim@startline
    \expandafter\verbatim@read@file
  \fi
}
\def\verbatiminput{\begingroup\MacroFont
  \@ifstar{\verbatim@input\relax}%
          {\verbatim@input{\frenchspacing\@vobeyspaces}}}
\def\verbatim@input#1#2{%
   \IfFileExists {#2}{\@verbatim #1\relax
    \verbatim@readfile{\@filef@und}\endtrivlist\endgroup\@doendpe}%
   {\typeout {No file #2.}\endgroup}}
⟨/strukdoc⟩
```

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

```
⟨*makefile⟩
#-----------------------------------------------------------------------
# Purpose: generation of the documentation of the struktex package
# Notice:  this file can be used only with dmake and the option "-B";
#          this option lets dmake interpret the leading spaces as
#          distinguishing characters for commands in the make rules.
#
# Rules:
#          - all-de:     generate all the files and the (basic) german
#                        documentation
#          - all-en:     generate all the files and the (basic) english
#                        documentation
#          - test:       format the examples
#          - history:    generate the documentation with revision
#                        history
#          - develop-de: generate the german documentation with revision
#                        history and source code
#          - develop-en: generate the english documentation with
#                        revision history and source code
#          - realclean
#          - clean
#          - clean-example
#
# Author:  Jobst Hoffmann, Fachhochschule Aachen, Standort Juelich
# Date:    2017/06/06
#-----------------------------------------------------------------------

# The texmf-directory, where to install new stuff (see texmf.cnf)
# If you don't know what to do, search for directory texmf at /usr.
# With teTeX and linux often one of following is used:
#INSTALLTEXMF=/usr/TeX/texmf
#INSTALLTEXMF=/usr/local/TeX/texmf
#INSTALLTEXMF=/usr/share/texmf
#INSTALLTEXMF=/usr/local/share/texmf
# user tree:
#INSTALLTEXMF=$(HOME)/texmf
# Try to use user's tree known by kpsewhich:
INSTALLTEXMF=`kpsewhich --expand-var '$$TEXMFHOME'`
# Try to use the local tree known by kpsewhich:
#INSTALLTEXMF=`kpsewhich --expand-var '$$TEXMFLOCAL'`
# But you may set INSTALLTEXMF to every directory you want.
# Use following, if you only want to test the installation:
#INSTALLTEXMF=/tmp/texmf

# If texhash must run after installation, you can invoke this:
TEXHASH=texhash

######### Edit following only, if you want to change defaults!

# The directory, where to install *.cls and *.sty
```

```
CLSDIR=$(INSTALLTEXMF)/tex/latex/$(PACKAGE)

# The directory, where to install documentation
DOCDIR=$(INSTALLTEXMF)/doc/latex/$(PACKAGE)

# The directory, where to install the sources
SRCDIR=$(INSTALLTEXMF)/source/latex/$(PACKAGE)

# The directory, where to install demo-files
# If we have some, we have to add following 2 lines to install rule:
#     $(MKDIR) $(DEMODIR); \
#     $(INSTALL) $(DEMO_FILES) $(DEMODIR); \
DEMODIR=$(DOCDIR)/demo

# We need this, because the documentation needs the classes and packages
# It's not really a good solution, but it's a working solution.
TEXINPUTS := $(PWD):$(TEXINPUTS)

########################################################################
#   End of customization section
########################################################################

LATEX = latex
PDFLATEX = pdflatex
TEX = TEX

COMMON_OPTIONS = # \OnlyDescription\CodelineNumbered\PageIndex
HISTORY_OPTIONS = \RecordChanges
DEVELOPER_OPTIONS = \EnableCrossrefs\RecordChanges\AlsoImplementation\CodelineIndex

# tarring options
EXgit = --exclude .git --exclude .gitignore --exclude auto --exclude tests \
 --exclude *.tgz --exclude *.bib

# The name of the game
PACKAGE = struktex

DISTRIBUTION_FILES = ../$(PACKAGE)/$(PACKAGE).de.pdf \
../$(PACKAGE)/$(PACKAGE).en.pdf \
 ../$(PACKAGE)/$(PACKAGE).dtx \
 ../$(PACKAGE)/$(PACKAGE).ins \
 ../$(PACKAGE)/LIESMICH.md \
 ../$(PACKAGE)/README.md
PACKAGE_FILES_A = $(subst ../$(PACKAGE)/,,$(DISTRIBUTION_FILES))
PACKAGE_FILES_B = $(subst $(PACKAGE).dtx ,,$(PACKAGE_FILES_A))
PACKAGE_FILES_C = $(subst $(PACKAGE).ins ,,$(PACKAGE_FILES_B))
PACKAGE_FILES_D = $(subst LIESMICH.md,,$(PACKAGE_FILES_C))
PACKAGE_FILES = $(subst README.md,,$(PACKAGE_FILES_D))

# To generate the version number of the distribution from the source
VERSION_L := git describe --long | xargs git --no-pager show -s \
         --date=short --format=format:"$(PACKAGE) version ???? of %ad%n" |\
         sed -e "s/????/‘git describe --long‘/"
VERSION_S := ‘git describe --long | \
                sed 's+-g.*++'`

# to create the correct tar-file
define TAR_COMMANDS
echo $$@
OUT_DIR=$$(mktemp -d)
mkdir $${OUT_DIR}/struktex
cp $$@ $${OUT_DIR}/struktex
pushd $${OUT_DIR}
tar cfvz struktex.tgz struktex
popd
cp $${OUT_DIR}/struktex.tgz .
endef

export TAR_COMMANDS

## Main Targets

# strip off the comments from the package
$(PACKAGE).sty $(PACKAGE)-test-*.tex: $(PACKAGE).ins $(PACKAGE).dtx $(PACKAGE).sed
 +$(TEX) $<; \
  source $(PACKAGE).makemake; \
  make revision_no; \
  source $(PACKAGE).sed  # set version number

all-de: $(PACKAGE).de.pdf

all-en: $(PACKAGE).en.pdf

# prepare the file with git revision information
.PHONY: revision_no
revision_no: $(PACKAGE).sed

$(PACKAGE).sed: $(PACKAGE).dtx
 printf "%b\n" "set_git_info() {" \
    > $(PACKAGE).sed; \
 printf "%b\n" "sed -i -e 's/^[ \\\\t]*%% git revision information$$/\\"  \
    >> $(PACKAGE).sed; \
 git describe --long | \
    xargs git --no-pager show -s --format=format:\
 "    \\\\@git@ \$$Date: %ci $$%%\%n" >> $(PACKAGE).sed; \
 git describe --long | cut -c 2- |\
    sed -e "s/^/      \$$Revision: /" -e "s/$$/ $$\\\\/" \
    >> $(PACKAGE).sed; \
 git describe --long | \
    xargs git --no-pager show -s --format=format:"    %%%% \$$Author: %an $$\\%n" \
    >> $(PACKAGE).sed; \
 printf "%b\n" "/' \$$1" \
    >> $(PACKAGE).sed; \
 printf "%b\n" "};" \
    >> $(PACKAGE).sed; \
 printf "%b\n" "for f in  \\\\" \
    >> $(PACKAGE).sed; \
 printf "%b\n" "$(PACKAGE).sty \\\\" \
    >> $(PACKAGE).sed; \
 printf "%b\n" "struktxf.sty \\\\" \
    >> $(PACKAGE).sed; \
 printf "%b\n" "struktxp.sty \\\\" \
    >> $(PACKAGE).sed; \
 printf "%b\n" "strukdoc.sty \\\\" \
    >> $(PACKAGE).sed; \
 printf "%b\n" "; do \\\\" \
    >> $(PACKAGE).sed; \
 printf "%b\n" "  set_git_info \$$f; done" \
    >> $(PACKAGE).sed; \

# generate the documentation
$(PACKAGE).de.pdf: $(PACKAGE).dtx $(PACKAGE).sty struktex.sed
 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{$<}"
 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{$<}"
 +mv $(<:.dtx=.pdf) $(<:.dtx=.de.pdf)

$(PACKAGE).en.pdf: $(PACKAGE).dtx $(PACKAGE).sty
 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\primarylanguage{english}\input{$<}"
 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\primarylanguage{english}\input{$<}"
 +mv $(<:.dtx=.pdf) $(<:.dtx=.en.pdf)

# generate the documentation with revision history (only german)
history: $(PACKAGE).dtx $(PACKAGE).sty
 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(HISTORY_OPTIONS)}\input{$<}"
 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(HISTORY_OPTIONS)}\input{$<}"
 +makeindex -s gind.ist                 $(PACKAGE).idx
 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(HISTORY_OPTIONS)}\input{$<}"

# generate the documentation for the developer (revision history always
# in german)
develop-de: $(PACKAGE).dtx $(PACKAGE).sty
 +$(PDFLATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)$(DEVELOPER_OPTIONS)}\input{$<}"
 +$(PDFLATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)$(DEVELOPER_OPTIONS)}\input{$<}"
 +makeindex -s gind.ist                 $(PACKAGE).idx
 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
 +$(PDFLATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)$(DEVELOPER_OPTIONS)}\input{$<}"
 +mv $(<:.dtx=.pdf) $(<:.dtx=.de.pdf)

develop-en: $(PACKAGE).dtx $(PACKAGE).sty
 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(DEVELOPER_OPTIONS)}\def\primarylanguage{eng
 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(DEVELOPER_OPTIONS)}\def\primarylanguage{eng
 +makeindex -s gind.ist                 $(PACKAGE).idx
 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(DEVELOPER_OPTIONS)}\def\primarylanguage{eng
 +mv $(<:.dtx=.pdf) $(<:.dtx=.en.pdf)

# format the example/test files
test:
 for i in `seq 1 3`; do \
     f=$(PACKAGE)-test-$$i; \
     echo file: $$f; \
     $(PDFLATEX) $$f; \
 done

install: $(PACKAGE).dtx $(PACKAGE).dvi
 [ -d $(CLSDIR) ] || mkdir -p $(CLSDIR)
 [ -d $(DOCDIR) ] || mkdir -p $(DOCDIR)
 [ -d $(SRCDIR) ] || mkdir -p $(SRCDIR)
 cp $(PACKAGE).sty      $(CLSDIR)
 cp $(PACKAGE).dvi      $(DOCDIR)
 cp $(PACKAGE).ins      $(SRCDIR)
 cp $(PACKAGE).dtx      $(SRCDIR)
 cp $(PACKAGE)-test-*.tex   $(SRCDIR)
 cp LIESMICH        $(SRCDIR)
 cp README          $(SRCDIR)
 cp THIS-IS-VERSION-$(VERSION)  $(SRCDIR)

uninstall:
 rm -f  $(CLSDIR)/$(PACKAGE).sty
 rm -fr $(DOCDIR)
 rm -fr $(SRCDIR)

dist: $(PACKAGE).de.pdf $(PACKAGE).en.pdf  $(PACKAGE).dtx  $(PACKAGE).ins \
LIESMICH.md README.md
 + echo "$$TAR_COMMANDS" > ./tar_commands; \
 rm -f THIS_IS_VERSION_*; \
 $(VERSION_L) > THIS_IS_VERSION_$(VERSION_S); \
 sh  ./tar_commands $^; \
 mv ./struktex.tgz ./struktex-$(VERSION_S).tgz
 rm ./tar_commands

tds-zip:  $(PACKAGE_FILES)
 + rm -f THIS_IS_VERSION_* *.zip; \
 $(VERSION_L) | sed -e "s/??????????/$(VERSION_S)/" > THIS_IS_VERSION_$(VERSION_S); \
 DOC_FILES="LIESMICH.md README.md THIS_IS_* $(PACKAGE).??.pdf"; \
 MAKE_FILES="$(PACKAGE).m*"; \
 SRC_FILES="$(PACKAGE).dtx $(PACKAGE).ins"; \
 STY_FILES="struk*.sty"; \
 TEST_FILES="./$(PACKAGE)-test*"; \
 SUPPORT_FILES="./$(PACKAGE).el"; \
 if [[ -d /tmp/texmf ]]; then \
   rm -rf /tmp/texmf; \
 fi; \
 if [[ -f $(PACKAGE)-TDS.zip ]]; then \
   rm $(PACKAGE)-TDS.zip; \
 fi; \
 mkdir -p /tmp/texmf/doc/latex/$(PACKAGE); \
 mkdir -p /tmp/texmf/source/latex/$(PACKAGE); \
 mkdir -p /tmp/texmf/tex/latex/$(PACKAGE); \
 cp -a $${DOC_FILES} /tmp/texmf/doc/latex/$(PACKAGE); \
 cp -a $${MAKE_FILES} /tmp/texmf/doc/latex/$(PACKAGE); \
 cp -a $${SRC_FILES} /tmp/texmf/source/latex/$(PACKAGE); \
 cp -a $${STY_FILES} /tmp/texmf/tex/latex/$(PACKAGE); \
 cp -a $${TEST_FILES} /tmp/texmf/doc/latex/$(PACKAGE); \
 cp -a $${SUPPORT_FILES} /tmp/texmf/doc/latex/$(PACKAGE); \
 VERSION_SHORT="xxx"; \
 pushd /tmp/texmf; \
 zip -r /tmp/$(PACKAGE)-TDS.zip .; \
 popd; \
 mv /tmp/$(PACKAGE)-TDS.zip ./$(PACKAGE)-TDS-$(VERSION_S).zip


clean:
 -rm -f *.log *.aux *.brf *.idx *.ilg *.ind
 -rm -f *.glg *.glo *.gls *.lof *.lot *.out *.toc *.tmp *~
 -rm *.mk *.makemake

realclean:  clean
 -rm -f *.sty *.cls *.pdf
 -rm -f *-test-* Makefile

clean-test:
 rm $(PACKAGE)-test-*.* # this $-sign is needed for font-locking in XEmacs only
⟨/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`.

```
⟨*setup⟩
sed -e "`echo \"s/^ /@/g\" | tr '@' '\011'`" struktex.mk > Makefile
⟨/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/LATEX files. If there is a suitable AUCTeX style file for a LATEX package like the hereby provided StrukTEX 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.

```
⟨*auctex⟩
;;; struktex.el --- AUCTeX style for `struktex.sty'

;; Copyright (C) 2006 - 2017 Free Software Foundation, Inc.

;; Author: J. Hoffmann <j.hoffmann_(at)_fh-aachen.de>
;; Maintainer: j.hoffmann_(at)_fh-aachen.de
;; Created: 2017/06/06
;; Keywords: tex
```

```

;;; Commentary:
;;  This file adds support for ‘struktex.sty’

;;; Code:

(TeX-add-style-hook
 "struktex"
 (lambda ()
   ;; Add declaration to the list of environments which have
   ;; an optional argument for each item.
   (add-to-list 'LaTeX-item-list
                '("declaration" . LaTeX-item-argument))
   (LaTeX-add-environments
    "centernss"
    '("struktogramm" LaTeX-env-struktogramm)
    '("declaration" LaTeX-env-declaration))
   (TeX-add-symbols
    '("PositionNSS" 1)
    '("assert" [ "Height" ] "Assertion")
    '("assign" [ "Height" ] "Statement")
    "StrukTeX"
    '("case" TeX-mac-case)
    "switch" "Condition"
    "caseend"
    '("declarationtitle" "Title")
    '("description" "Name" "Meaning")
    "emptyset"
    '("exit" [ "Height" ] "What" )
    '("forever" TeX-mac-forever)
    "foreverend"
    '("forallin" TeX-mac-forallin)
    "forallin"
    '("ifthenelse" TeX-mac-ifthenelse)
    "change"
    "ifend"
    '("inparallel" TeX-mac-inparallel)
    '("task" "Description")
    "inparallelend"
    "sProofOn"
    "sProofOff"
    '("until" TeX-mac-until)
    "untilend"
    '("while" TeX-mac-while)
    "whileend"
    '("return" [ "Height" ] "Return value")
    '("sub" [ "Height" ] "Task")
    '("CenterNssFile" TeX-arg-file)
    '("centernssfile" TeX-arg-file))
   (TeX-run-style-hooks
    "pict2e"
    "emlines2"
    "curves"
    "struktxp"
   "struktxf"
   "ifthen")
  ;; Filling
  ;; Fontification
  ))

(defun LaTeX-env-struktogramm (environment)
  "Insert ENVIRONMENT with width, height specifications and
optional title."
  (let ((width (read-string "Width: "))
        (height (read-string "Height: "))
        (title (read-string "Title (optional): ")))
    (LaTeX-insert-environment environment
                              (concat
                               (format "(%s,%s)" width height)
                               (if (not (zerop (length title)))
                                   (format "[%s]" title))))))

(defun LaTeX-env-declaration (environment)
  "Insert ENVIRONMENT with an optional title."
  (let ((title (read-string "Title (optional): ")))
    (LaTeX-insert-environment environment
                              (if (not (zerop (length title)))
                                  (format "[%s]" title)))))

(defun TeX-mac-case (macro)
  "Insert \\case with all arguments, the needed \\switch(es) and
the final \\caseend. These are optional height and the required
arguments slope, number of cases, condition, and the texts for
the different cases"
  (let ((height (read-string "Height (optional): "))
        (slope (read-string "Slope: "))
        (number (read-string "Number of cases: "))
        (condition (read-string "Condition: "))
        (text (read-string "Case no. 1: "))
        (count 1)
        )
    (setq number-int (string-to-number number))
    (insert (concat (if (not (zerop (length height)))
                        (format "[%s]" height))
                    (format "{%s}{%s}{%s}{%s}"
                            slope number condition text)))
    (while (< count number-int)
      (end-of-line)
      (newline-and-indent)
      (newline-and-indent)
      (setq prompt (format "Case no. %d: " (+ 1 count)))
      (insert (format "\\switch{%s}" (read-string prompt)))
      (setq count (1+ count)))
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\caseend")))

```

```
(defun TeX-mac-forallin (macro)
  "Insert \\forallin-block with all arguments.
This is the optional height and the description of the loop"
  (let ((height (read-string "Height (optional): "))
        (loop-description (read-string "Description of loop: ")))
    (insert (concat (if (not (zerop (length height)))
                        (format "[%s]" height))
                    (format "{%s}"
                            loop-description)))
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\forallinend")))

(defun TeX-mac-forever (macro)
  "Insert \\forever-block with all arguments.
This is only the optional height"
  (let ((height (read-string "Height (optional): ")))
    (insert (if (not (zerop (length height)))
                (format "[%s]" height)))
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\foreverend")))

(defun TeX-mac-ifthenelse (macro)
  "Insert \\ifthenelse with all arguments.
These are optional height and the required arguments
left slope, right slope, condition, and the possible
values of the condition"
  (let ((height (read-string "Height (optional): "))
        (lslope (read-string "Left slope: "))
        (rslope (read-string "Right slope: "))
        (condition (read-string "Condition: "))
        (conditionvl (read-string "Condition value left: "))
        (conditionvr (read-string "Condition value right: ")))
    (insert (concat (if (not (zerop (length height)))
                        (format "[%s]" height))
                    (format "{%s}{%s}{%s}{%s}{%s}"
                            lslope rslope condition conditionvl
                            conditionvr)))
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\change")
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\ifend")))

(defun TeX-mac-inparallel (macro)
  "Insert \\inparallel with all arguments, the needed \\task(s)
and the final \\inparallelend. These are optional height and the
required arguments number of tasks and the descriptions for the
parallel tasks"
  (let ((height (read-string "Height (optional): "))
        (number (read-string "Number of parallel tasks: "))
        (text (read-string "Task no. 1: "))
        (count 1)
        )
    (setq number-int (string-to-number number))
    (insert (concat (if (not (zerop (length height)))
                        (format "[%s]" height))
                    (format "{%s}{%s}" number text)))
    (while (< count number-int)
      (end-of-line)
      (newline-and-indent)
      (newline-and-indent)
      (setq prompt (format "Task no. %d: " (+ 1 count)))
      (insert (format "\\task{%s}" (read-string prompt)))
      (setq count (1+ count)))
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\inparallelend")))

(defun TeX-mac-until (macro)
  "Insert \\until with all arguments.
These are the optional height and the required argument condition"
  (let ((height (read-string "Height (optional): "))
        (condition (read-string "Condition: ")))
    (insert (concat (if (not (zerop (length height)))
                        (format "[%s]" height))
                    (format "{%s}" condition)))
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\untilend")))

(defun TeX-mac-while (macro)
  "Insert \\while with all arguments.
These are the optional height and the required argument condition"
  (let ((height (read-string "Height (optional): "))
        (condition (read-string "Condition: ")))
    (insert (concat (if (not (zerop (length height)))
                        (format "[%s]" height))
                    (format "{-%s-}" condition)))
    (end-of-line)
    (newline-and-indent)
    (newline-and-indent)
    (insert "\\whileend")))

(defvar LaTeX-struktex-package-options '("curves" "draft" "emlines" "final"
                                         "pict2e" "anygradient" "verification"
                                         "nofiller" "debug" "outer")
  "Package options for the struktex package.")

;;; struktex.el ends here.
```

1221 ⟨/auctex⟩

## References

[Fut89] Gerald Futschek. *Programmentwicklung und Verifikation.* Springer Verlag, Wien – New York, 1989.

[GMS94] Michel Goossens, Frank Mittelbach and Alexander Samarin. *The LaTeX-Companion.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.

[GMS04] Frank Mittelbach and Michel Goossens. *The LaTeX-Companion.* Addison-Wesley Publishing Company, Reading, Massachusetts, second edition, 2004.

[Knu86] D. E. Knuth. *The TeX-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 TeXnische Kom"odie*, 8(4):23–40, Februar 1996.

[Rah92] Sebastian Rahtz. *The* oz *package*, 1992.