struktex.sty*

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

printed on June 20, 2025

Abstract

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

Contents

1 License **1**

2 Preface **2**

3 Hints for maintenance and installation of this documentation **3**

4 The User interface **5**
4.1 The macros for generating structured box charts 6
4.2 Specific characters and text representation 20
4.3 Macros for representing variables, keywords and other programming-specific details 21
4.4 Macros for generating the documentation of the package struktex.sty 23

5 Several sample files **28**
5.1 Testing the package struktxf.sty 28
5.2 File for testing the package struktxp.sty 29
5.3 Positioning of structured box charts with \centernss/\input . . . 33
5.4 Predefined slopes at selections 34
5.5 Variable slopes at selections 35
5.6 Documentation of Java programs 36

6 Checking and building the package struktex.sty **37**

7 Style file for easier input while working with the Emacs and AUCTeX **42**

1 License

*This file has version number v3.0a-2-gbbdacc3, last revised on 2025/06/20, documentation dated 2025/01/25.

Jobst Hoffmann, 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 $\mathcal{AMS}$-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. specially 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 slopes. 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.

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] 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.

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_ε 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 of this documentation

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

`LIESMICH.md`,
`README.md`,
`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.dtx will be formatted e.g. with

```
pdftex struktex.dtx
```

This formatting run creates a few more files. These are first of all the three .sty-files struktex.sty, struktxf.sty and struktxp.sty, the latter two are used by struktex.sty. Furthermore these are the two files struktex-tst-00.nss and strukdoc.sty, which are used for the generation of the hereby presented documentation. Then there are five test files struktex-test-*i*.tex, $i = 1(1)3$, struktxf.tex, and struktxf.tex for testing the corresponding .sty-files and last, but not least two script files lbuild.lua and lbuild-config.lua (see section 6).

The documentation is generated according to l3build with the command

```
l3build doc
```

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 6), 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 shouldn't begin with "\" (*backslash*), but with the "‘" (*accent*).

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

```
%%
%% This is file ‘struktex.drv’,
%% generated with the docstrip utility.
%%
%% The original source files were:
%%
%% struktex.dtx  (with options: ‘driver’)
%%
%% Copyright (C) 1989-2025 by Jobst Hoffmann.  All rights reserved.
%%
%% IMPORTANT COPYRIGHT NOTICE:
%%
%% No other permissions to copy or distribute this file in any form
%% are granted and in particular NO PERMISSION to modify its contents.
%%
%% You are NOT ALLOWED to change this file.
%%
%% Please address error reports and any problems in case of UNCHANGED versions
%% to
%%         j.hoffmann_(at)_fh-aachen.de
                                  % set the default formatting language:
\expandafter\ifx\csname primarylanguage\endcsname\relax
    \def\primarylanguage{ngerman}
    \def\secondarylanguage{english}
\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{moreverb}             % for formatting the documentation of the driver
\usepackage{strukdoc}             % the style-file for formatting this
                                  % documentation
```

```
37
38  \usepackage[pict2e, % <---------- to produce finer results
39                                  % visible under xdvi, alternatives are
40                                  % curves or emlines2 (visible only under
41                                  % ghostscript), leave out if not
42                                  % available
43      verification,
44      outer, % <------------------- to set the position of the \ifthenelse
45                                  % flags to the outer edges
46      debug,
47      ]
48      {struktex}
49
50  \OnlyDescription    % add comment char to include the implementation details
51
52  \MakeShortVerb{\|}    % |\foo| acts like \verb+\foo+
53
54  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
55  %% to avoid underfull ... messages while formatting two/three columns
56  \hbadness=10000 \vbadness=10000
57
58  \typeout{\string\primarylanguage: \primarylanguage, \string\language: \the\language}
59  \def\languageNGerman{\the\csname l@ngerman\endcsname} % depends on language.dat,
60                                  % there may be changes from release to
61                                  % release
62
63  \begin{document}
64  \makeatletter
65  \@ifundefined{selectlanguageEnglish}{}{\selectlanguage{english}}
66  \makeatother
67  \DocInput{struktex.dtx}
68  \end{document}
69
70  \endinput
71  %%
72  %% End of file `struktex.drv'.
```

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 slope 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 The macros for generating structured box charts

`struktogramm` (*env.*) `\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.

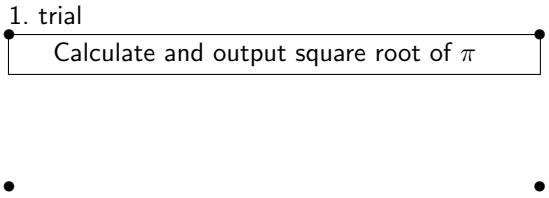
Example 1

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

```
\sProofOn
\begin{struktogramm}(70,20)[1.\ trial]
    \assign{Calculate and output square root of \(\pi\)}
\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 18

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:

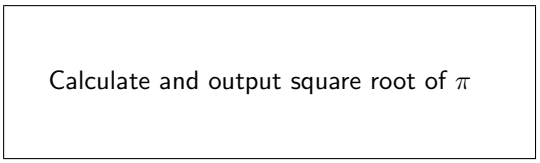
Example 2

The height of the box is given by:

```
\begin{center}
\begin{struktogramm}(70,20)
    \assign[20]{Calculate and output square root of \(\pi\)}
\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` (*env.*) 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⟩}
```

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 StruktTEX). 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}
```

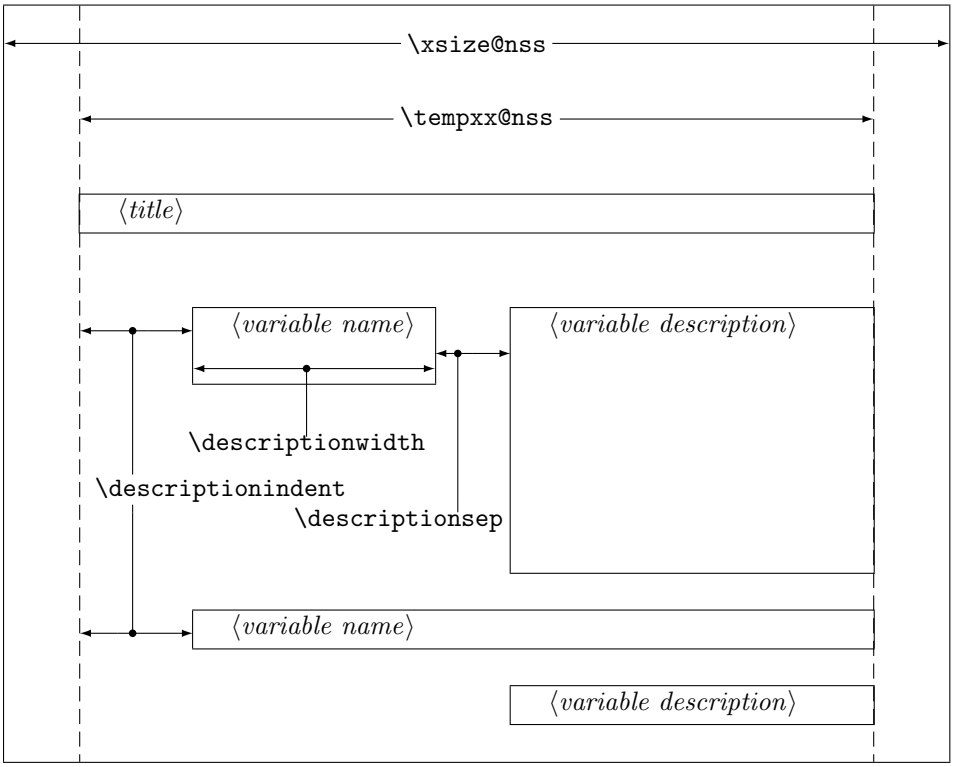


Figure 1: Construction of a variable description

```
        \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 described 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:

global variables
iVar_g {an *int* variable}

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*⟩}

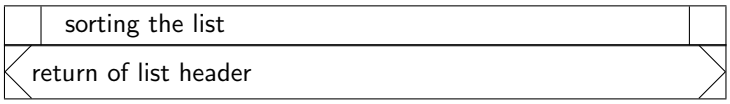
\return[⟨*height*⟩]{⟨*text*⟩}

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*⟩`\forallinend`

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`.

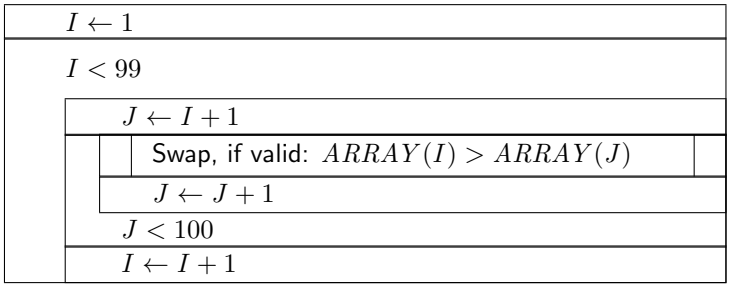
The two following examples show use of `\while` and `\until` macros. `\forever` will be shown later.

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}
```

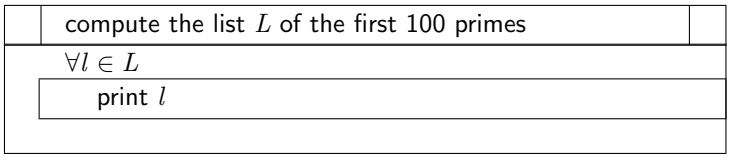
These instructions lead to the following structured box chart:



Example 6

```
\begin{struktogramm}(95, 25)
    \sub{compute the list \(L\) 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.

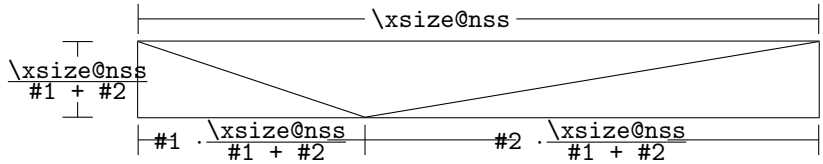
`\ifthenelse` `\change` `\ifend` For the representation of alternatives StruktTEX provides mapping conventions for an If-Then-Else-block and a Case-construction for multiple alternatives. Since in the traditional `picture` environment of LaTeX only lines of certain slopes can be drawn, in both cases the user has to specify himself the angle, with which the necessary slanted lines shall be drawn. (Here is a little bit more 'manual work' required.)

If however the curves.sty, the emlines2.sty or the pict2e.sty is used, then the representation of lines with any slope 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 slope of both the partitioning lines of the If-Then-Else-block (large number = small slope). Larger values are put on 6, smaller values on 1. The precise characteristics of the slopes 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 $\frac{\xsize@nss}{\#1 + \#2}$.



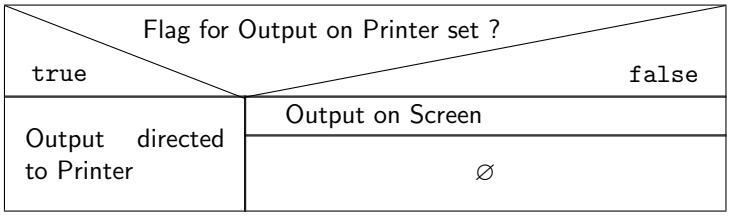
⟨*condition*⟩ is set in the upper triangle built in the above way. The parameters ⟨*left text*⟩ and ⟨*right text*⟩ are set in the left or right lower triangle respectively. The conditioning text can be made up in its triangle box. As of version v-5.3, the condition text is adapted to any slopes using suitable wrapping.[2] 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` (see section 4.3) 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 ∅ (see section 4.2) 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:

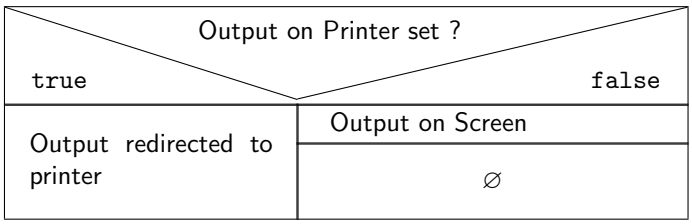
[2] This extension is due to Daniel Hagedorn, whom I have to thank for his work.



Example 8

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

These instructions lead to the following structured box chart:



`\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*⟩

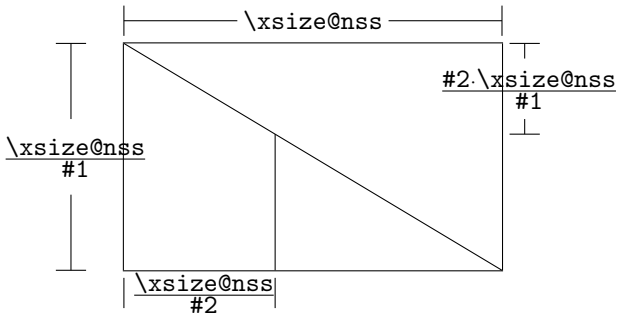
...

`\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 slope 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:

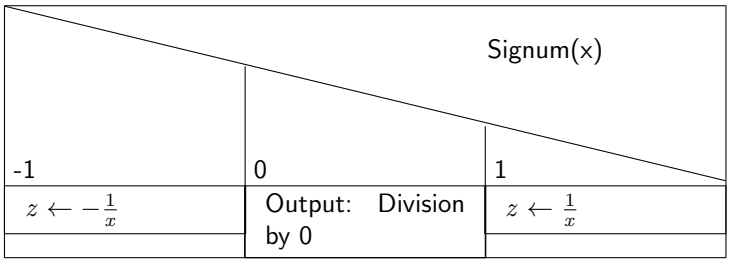


The second parameter ⟨*number of cases*⟩ specifies the number of cases, that have to be drawn. All structured subbox charts of the certain cases get the same width. The ⟨*text of 1. case*⟩ has to be given as a parameter of the `\case` instruction. All other cases are introduced by the `\switch` instruction.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

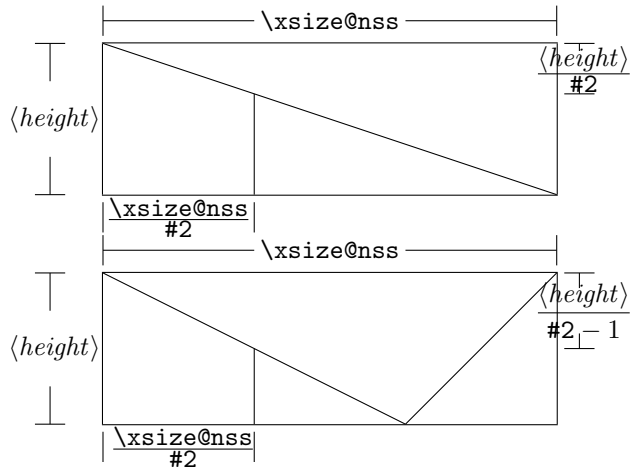
Example 9

```
\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 scrumbled up. The extension of the `\switch` instruction by `[`⟨*height*⟩`]` results in the following shape with a different slope 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:

Signum(x)		
-1	0	1
$z \leftarrow -\frac{1}{x}$	Output: Division by 0	$z \leftarrow \frac{1}{x}$

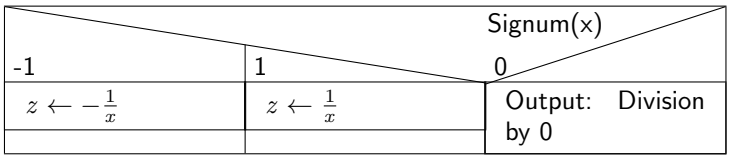
But if the first parameter is odd, then a default branch is drawn; the value for the default branch should be set flushed right.

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}
```

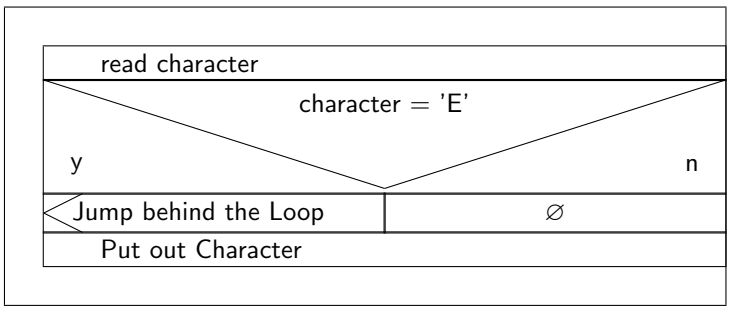
These instructions lead to the following structured box chart:



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:

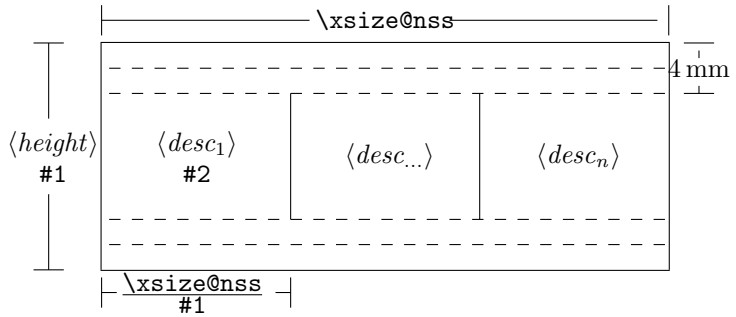
`\inparallel[`⟨*height of 1st task*⟩`]{`⟨*number of parallel tasks*⟩`}{`⟨*description of 1st task*⟩`}}`
`\task[`⟨*position*⟩`]{`⟨*description of 2nd task*⟩`}`

...

`\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`):

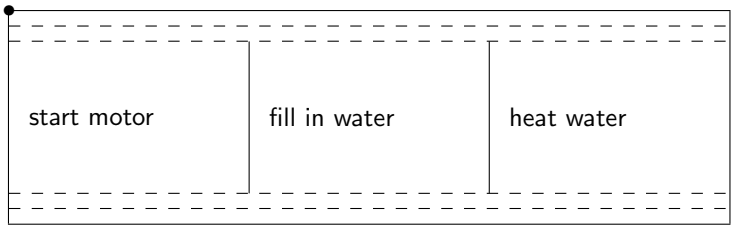


Note: the tasks are not allowed to get divided by `\assign` or so. If one needs some finer description of a task, this should be made outside of the current structured box chart.

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` (*env.*) If a structured box chart shall be represented centered, then the environment

`\begin{centernss}`

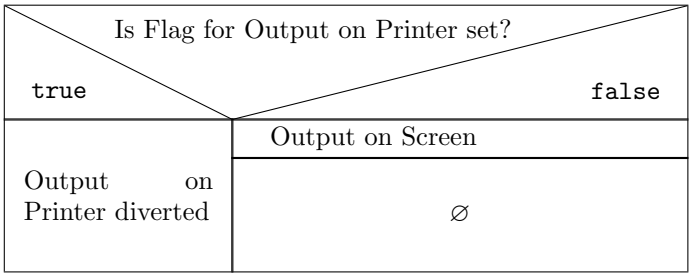
⟨*Struktogramm*⟩

`\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:



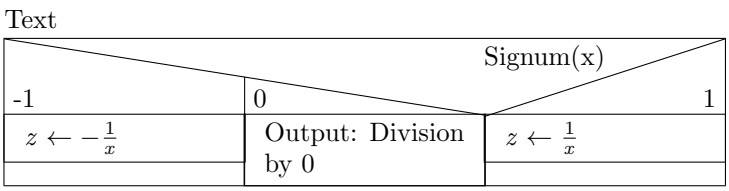
`\CenterNssFile` In many cases structured box charts are recorded in particular files such, that they can be tested seperately, if they are correct, or that they can be used in other connections. If they should be included centeredly, then one can not use the following construction:

```
\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-tst-00.nss has the shape shown in section 5.3 on page 33, line 61–68 the instruction

```
\centernssfile{struktex-tst-00}
```

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 StruktTEX. Their meaning is the same as \struktogramm and \endstruktogramm. The syntax is

\openstrukt{⟨*width* ⟩}{⟨*height* ⟩}

and

\closestrukt.

\assert

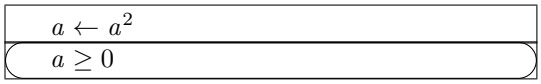
The macro \assert was introduced to support the verification of algorithms. It is active only if the option verification is set. It serves the purpose to assert the value of a variable at one point of the algorithm. The syntax corresponds to the syntax of \assign:

\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



4.2 Specific characters and text representation

\nat
\integer
\real
\complex
\emptyset

Since sets of natural, whole, real and complex numbers (ℕ, ℤ, ℝ and 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 "∅". Other set symbols like 𝕃 (for solution space) have to be generated by \(\mathbb L\).

\MathItalics
\MathNormal

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

$$NewValue = OldValue + Correction \tag{1}$$

```
\MathNormal
\begin{equation}
    NewValue = OldValue + Correction
    \label{eq:-neuerwert-N}
\end{equation}
```

and

$$NewValue = OldValue + Correction \tag{2}$$

```
\MathItalics
\begin{equation}
    NewValue = OldValue + Correction
    \label{eq:-neuerwert-I}
\end{equation}
```

These commands are defined in a separate package struktxf.sty, which can be loaded as usual:

```
\usepackage[⟨options⟩]{struktxf}
```

The following options are available:

1. `iso`:

   This option specifies the representation of numerical quantities in accordance with ISO 80 000-2. If it is set, the numerical quantities are represented by bold capital letters such as $\mathbf{Q}$, otherwise by characters from the `\mathbb` character set such as $\mathbb{Q}$. The default value is `iso=false`.

2. `mathitalics`:

   Setting this option results in assignments, such as those occurring in of the description of algorithms, being displayed as in **??** on page ?? instead of **??** on page ??.

   It should be noted that this option is incompatible with the commands `\boldsymbol{...}` (from amsmath.sty) or `\bm{...}` (from bm.sty) and therefore `\MathNormal` must be activated before these commands.

4.3 Macros for representing variables, keywords and other programming-specific details

`\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 ampersand "&" 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 number sign "#" 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` Boolean values play an importand role in programming. There are given adequate values by `\pTrue` and `\pFalse`: `true` and `false`.

`\pFonts` `\pBoolValue` 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.

4.4 Macros for generating the documentation of the package 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}{}
\RequirePackage{varioref}
\ifpdf
    \RequirePackage[colorlinks]{hyperref}
\else
    \def\href#1{\texttt}
\fi
\RequirePackage{cleveref}
\ifcolor
    \RequirePackage{color}
\fi
\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}}
\newcommand\sFile[1]{\textsf{#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⟩
```

5 Several sample files

5.1 File for testing the macros of the package **struktxf.sty**

```
⟨*struktxf-00⟩
\documentclass[fleqn, 12pt, english]{article} % 12pt for better readability

\usepackage{babel}
\usepackage[iso, mathitalics]{struktxf} % the options could be omitted,
                                        % they represent the default

\nofiles

\begin{document}

\section*{Mathematical Sets:}

\begin{itemize}
  \item natural numbers: \(\nat\)
  \item integer numbers: \(\integer\)
  \item rational numbers: \(\rational\)
  \item real numbers: \(\real\)
  \item complex numbers: \(\complex\)
\end{itemize}
To compare the fonts visually, should not be done in real documents:
\begin{verbatim}
\makeatletter\@nss@isofalse\makeatother
\end{verbatim}
\makeatletter\@nss@isofalse\makeatother
\begin{itemize}
  \item natural numbers: \(\nat\)
  \item integer numbers: \(\integer\)
  \item rational numbers: \(\rational\)
  \item real numbers: \(\real\)
  \item complex numbers: \(\complex\)
\end{itemize}
or (with \verb|\@nss@isotrue|)
\makeatletter\@nss@isotrue\makeatother
\begin{itemize}
  \item natural numbers: \(\nat\)
  \item integer numbers: \(\integer\)
  \item rational numbers: \(\rational\)
  \item real numbers: \(\real\)
  \item complex numbers: \(\complex\)
\end{itemize}

\section*{Assignments:}

```

```
\begin{itemize}
  \item Default settings:
    \begin{equation}
        \label{eq:def}
        NewValue = OldValue + Correction
    \end{equation}
  \item \verb|MathNormal| settings: \MathNormal
    \begin{equation}
        \label{eq:norm}
        NewValue = OldValue + Correction
    \end{equation}
  \item \verb|MathItalics| settings: \MathItalics
    \begin{equation}
        \label{eq:ital}
        NewValue = OldValue + Correction
    \end{equation}
\end{itemize}

\end{document}
⟨/struktxf-00⟩
```

5.2 File for testing the macros of the package **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}`.

```
⟨*example3⟩
\documentclass[english]{article}

\usepackage{babel}
\usepackage{struktxf}
\usepackage{struktxp}

\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
}
```

```
339
340 \paragraph{after changing the fonts with}
341 \verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:
342
343 {\obeylines
344 \pFonts{\itshape}{\sffamily\bfseries}{}
345 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
346 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
347 in math mode: \(\pVar{a}+\pVar{iV_g}\)
348 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
349 }
350
351 \paragraph{after changing the fonts with}
352 \verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:
353
354 {\obeylines
355 \pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
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{\itshape}{\bfseries\itshape}{}-:
364
365 {\obeylines
366 \pFonts{\itshape}{\bfseries\itshape}{}
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 \vspace{15pt}
373 Without \textit{italic correction}:
374     M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
375 }
376
377 \pLanguage{C}
378 \pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
379 \section*{Default values (C):}
380
381 {\obeylines
382 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
383 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
384 in math mode: \(\pVar{a}+\pVar{iV_g}\)
385 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
386 }
387
388 \paragraph{After changing the boolean values with}
389 \verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:
390
391 {\obeylines
392 \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}
⟨/example3⟩
```

5.3 Sample file for including into the documentation: Positioning of structured box charts with `\centernss`/`\input`

The following lines build up a sample file, which is needed for the preparation of this documentation (see p. 18); there is only a german version of this file.

⟨∗nss-sample⟩

```
\begin{struktogramm}(95,40)[Text]
    \case[10]{3}{3}{Signum(x)}{-1}
         \assign{\(z \gets - \frac{1}{x}\)}
    \switch{0}
         \ifnum\language=\languageNGerman%
             \assign{Ausgabe: Division durch 0}
         \else
             \assign{Output: Division by 0}
         \fi
    \switch[r]{1}
         \assign{\(z \gets \frac{1}{x}\)}
    \caseend
\end{struktogramm}
⟨/nss-sample⟩
```

5.4 Display of selections with predefined slopes without the use of optional packages

The following lines build up a sample file, that can be used for testing the macros. The slope used at the `\ifthenelse` macro is one of the six cases defined in LaTeX 2_ε, because no optional package is loaded.

```
⟨*example1⟩
\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}
⟨/example1⟩
```

5.5 Display of case distinctions with variable slopes with the package **pict2e.sty**

The following lines build up a template file, that can be used for testing the macros. It is shown, how the slope gets automatically calculated after loading the package pict2e.

```
⟨*example2⟩
\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}
```
⟨/example2⟩

5.6 Documentation of Java methods in the header of a structure chart with the package **struktxp.sty**

The following code serves as a sample for documenting Java methods. Here a special method is used, because the common use of `\lstinline` from the listings package leads to errors.

The code also shows how the documented methods can be listed in the index.

⟨*struktxp-00⟩
```
\documentclass{article}

\usepackage{index} % produce the index afterwards with the command
                   % makeindex -o struktxp-tst-00.ind struktxp-tst-00.idx
\usepackage{struktxp}

\makeindex

\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{Methods!drawImage(Image img,
                                    int dx1,
                                    int dy1,
                                    int dx2,
                                    int dy2,
                                    int sx1,
                                    int sy1,
                                    int sx2,
                                    int sy2,
                                    ImageObserver observer)@%
       \pExp{\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}

\printindex
\end{document}
⟨/struktxp-00⟩
```

6 Checking and building the package struktex.sty with l3build

Working with .dtx packages is much easier when working with a well suited tool like l3build which can be used on all platforms [LaT20].

For a special project there must exist a build.lua script to adapt the behaviour

to its needs. This script is accomplished by a settings file build-config.lua. Both files are integrated in this documentation, their usage isn't described here, it can be read in [LaT20].

```
⟨*lbuild⟩
#!/usr/bin/env texlua

-- Build script for "StrukTeX" files

-- Identify the bundle and module: the module may be empty in the case where
-- there is no subdivision
bundle = ""
module = "struktex"

-- Location of main directory: use Unix-style path separators
-- Should match that defined by the bundle.
maindir = "."

-- the files to be installed
installfiles = {"*.sty"}

-- the files which should not be installed
excludefiles = {"*/TODO.md"}

-- these files are to copy for unpackaging
sourcefiles = {"*.dtx", "*.ins", "build.lua", "build-config.lua"}

-- these files are part of the documentation
docfiles = {"struktex.drv", "struktex.sty"}

-- these files get tags
tagfiles = {"*.sty"}

-- this file must be TeXed to unpack all
unpackfiles = {"struktex.dtx"}
-- unpackopts = "-interaction=batchmode"

-- create a TDS-suited .zip-file
packtdszip = true

-- Files to copy to unpacking when typesetting
typesetsourcefiles = {"struktex.sty"}

-- Files needed to support typesetting when "sandboxed"?
typesetsuppfiles = {"tex-bib.bib"}

-- only using the following engines for testing the code
checkengines    = checkengines or {"latex", "pdftex"}

-- install also documentation (doesn't work, because
-- there are no *.pdf-files in the unpacked directory
installfiles     = installfiles or {"*.pdf", "*.sty"}

-- Load the common build code
dofile(maindir .. "/build-config.lua")
```

```
781
782 -- specific settings for the StrukTeX development repo, used by l3build script
783
784 function prepare_contents(file)
785   -- can't be set locally, so: set it back later on!
786   typesetopts = "--output-format=dvi"
787   tex(file)
788
789   -- create .pdf-file from .dvi-source
790   name = stripext(file)
791   dvitopdf(name, typesetdir, typesetexe, true)
792   run(typesetdir, "pdfcrop " .. name .. pdfext)
793   ren(typesetdir, name .. "-crop" .. pdfext, name .. pdfext)
794 end
795
796 function print_log(log_message)
797   print(log_message)
798   return 0
799 end
800
801 function typeset(file)
802   local name = jobname(file)
803
804   typesetopts = "--output-format=pdf"
805   local errorlevel = tex(file, typesetdir)
806
807   print("\ntypeset.errorlevel:", errorlevel, "\n")
808
809   if errorlevel == 0 then
810     -- Return a non-zero errorlevel if anything goes wrong
811     print("prepare additional information")
812     errorlevel = (
813       print_log("file: " .. file .. ", name: " .. name) +
814         print_log("going to bibtex with >" ..  name .. "<") +
815         bibtex(name, typesetdir) +
816         print_log("\nbibtex done\n") +
817         tex(file, typesetdir) +
818         print_log("going to makeindex with", name) +
819         makeindex(name, typesetdir, ".idx", ".ind", ".ilg", "gind.ist") +
820         makeindex(name, typesetdir, ".glo", ".gls", ".glg", "gglo.ist") +
821         print_log("\nmakeindex done\n") +
822         tex(file, typesetdir)
823                 )
824     print_log("\ntypesetting done\n")
825     ren(typesetdir, "struktex.pdf", "struktex." .. targetlanguage .. ".pdf")
826     print_log("\nrenaming to struktex." .. targetlanguage .. ".pdf done\n")
827   end
828
829   return errorlevel
830 end
831
832 function setversion_update_line(line, date, release)
833   local function get_date_release()
834     -- get version information from git
```

```
    local f = io.popen("git describe", "r")
    local release = {}
    for entry in f:lines() do
      release[#release + 1] = entry
    end
    return os.date("%Y/%m/%d", os.time()), release[1]:sub(2, -1)
  end

  date, release = get_date_release()

  for _,i in pairs({"Class", "File", "Package"}) do
    if string.match(
      line,
      "^\\ProvidesExpl" .. i .. " *{[a-zA-Z0-9%-%.]+}"
    ) then
      line = string.gsub(
        line,
        "{%d%d%d%d/%d%d/%d%d}( *){[^}]*}",
        "{" .. string.gsub(date, "%-", "/") .. "}%1{" .. release .. "}"
      )
      break
    end
  end
  return line
end

--###  doc_init_hook begin  ##############################################
--[[
  The git informations (commit name, commit date, commit author)
  must be included in the .sty files. This is done here.
  1. git describe --long HEAD: returns the most recent tag (HEAD
       can be omitted, it's the default), that's the commit name
  2. git --no-pager show -s --format=format:"<format>": returns
       the needed commit date and commit author. <format> is set
       up so that the three lines below are generated

  These informations replace the line
  %% git revision information
  by someting like
  \@git@ $Date: <commit date> $%
     $Revision: <commit name> $
  %% $Author: <commit author> $

  <commit date> has the format "yyyy-mm-dd hh:mm:ss +nnnn"
  <commit name> has the format as described in the git describe
    man page
--]]
function docinit_hook()
  print("now set git data")
  local f = io.popen('git describe --long HEAD', 'r')
  local commit_name = f:read "*all"
  commit_name = string.sub(commit_name, 1, #commit_name - 1)

  f = io.popen('git --no-pager show -s --format=format:"%n  ' ..
```

```
                '\\@git@ \\$Date: %ci \\$%%%%%n  \\$Revision: ' ..
                '<commit name> \\$%n  %%%%%%% \\$Author: %cn \\$%n"')
  local revision_information = f:read "*all"
  revision_information = string.sub(revision_information, 1,
      #revision_information - 1)

  print("commit_name: >>" .. commit_name .. "<<, Länge: " ..
      tostring(string.len(commit_name)))
  print("revision_information: " .. revision_information)

  revision_information = revision_information:gsub("<commit name>", commit_name)
  print("revision_information: " .. revision_information)

  filenames = {"strukdoc.sty", "struktex.sty", "struktxf.sty", "struktxp.sty"}

  for _, file in ipairs(filenames) do
    file = unpackdir .. "/" .. file
    print(file)

    local f = assert(io.open(file,"rb"))
    local content = f:read("*all")
    f:close()

    content = content:gsub("%%% git revision information",
                           revision_information)
    print(content)

    f = assert(io.open(file,"w"))
    f:write(content)
    f:close()

    if file == unpackdir .."/" .. "struktex.sty" then
      file = file:gsub(unpackdir, typesetdir)
      f = assert(io.open(file,"w"))
      f:write(content)
      f:close()
    end
  end

  -- this file is needed for the documentation but isn't automatically
  -- copied
  cp("struktex-tst-00.nss", unpackdir, typesetdir)

  return 0
end
--###  doc_init_hook end  ###############################################

-- Find and run the build system
kpse.set_program_name("kpsewhich")
if not release_date then
  dofile(kpse.lookup("l3build.lua"))
end

⟨/lbuild⟩
```

The file |build-config.lua consists of the following lines and is generated by executing the command `tex struktex.ins`. Its purpose is the switching between English and german documentation.

```
⟨*config⟩
-- configuration file for the StrukTeX package

```

"`\z`" allows breaking strings.

```
print("This is the additional configuration for setting \z
    the documentation language")

```

At this point the language of the documentation can be configured: one can choose between "de" and "en". `targetlanguage` must be global, because the value is referenced at the end of the documentation step.

```
targetlanguage = targetlanguage or "de"
print("target language: >" .. targetlanguage .. "<")

local exit = os.exit

if targetlanguage == "de" then
  typesetcmds = "\\def\\primarylanguage{ngerman}" ..
    "\\def\\secondarylanguage{english}"
elseif targetlanguage == "en" then
  typesetcmds = "\\def\\primarylanguage{english}" ..
    "\\def\\secondarylanguage{ngerman}"
else
  print("Illegal language")
  exit(1)
end
⟨/config⟩
```

7 Style file for easier input while working with the 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 StruckTEX 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 -*-coding: utf-8; lexical-binding: t-*-
;;; AUCTeX style for ‘struktex.sty’

;; Copyright (C) 2006 - 2025 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)
```

```
1081     (insert "\\caseend")))
1082
1083 (defun TeX-mac-forallin (macro)
1084   "Insert \\forallin-block with all arguments.
1085 This is the optional height and the description of the loop"
1086   (let ((height (read-string "Height (optional): "))
1087         (loop-description (read-string "Description of loop: ")))
1088     (insert (concat (if (not (zerop (length height)))
1089                         (format "[%s]" height))
1090                     (format "{%s}"
1091                             loop-description)))
1092     (end-of-line)
1093     (newline-and-indent)
1094     (newline-and-indent)
1095     (insert "\\forallinend")))
1096
1097 (defun TeX-mac-forever (macro)
1098   "Insert \\forever-block with all arguments.
1099 This is only the optional height"
1100   (let ((height (read-string "Height (optional): ")))
1101     (insert (if (not (zerop (length height)))
1102                 (format "[%s]" height)))
1103     (end-of-line)
1104     (newline-and-indent)
1105     (newline-and-indent)
1106     (insert "\\foreverend")))
1107
1108 (defun TeX-mac-ifthenelse (macro)
1109   "Insert \\ifthenelse with all arguments.
1110 These are optional height and the required arguments
1111 left slope, right slope, condition, and the possible
1112 values of the condition"
1113   (let ((height (read-string "Height (optional): "))
1114         (lslope (read-string "Left slope: "))
1115         (rslope (read-string "Right slope: "))
1116         (condition (read-string "Condition: "))
1117         (conditionvl (read-string "Condition value left: "))
1118         (conditionvr (read-string "Condition value right: ")))
1119     (insert (concat (if (not (zerop (length height)))
1120                         (format "[%s]" height))
1121                     (format "{%s}{%s}{%s}{%s}{%s}"
1122                             lslope rslope condition conditionvl
1123                             conditionvr)))
1124     (end-of-line)
1125     (newline-and-indent)
1126     (newline-and-indent)
1127     (insert "\\change")
1128     (end-of-line)
1129     (newline-and-indent)
1130     (newline-and-indent)
1131     (insert "\\ifend")))
1132
1133 (defun TeX-mac-inparallel (macro)
1134   "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" "debug"
                                         "draft" "emlines" "final"
                                         "fixedindent" "nofiller"
                                         "outer" "pict2e" "verification")
```

```
1189   "Package options for the struktex package.")
1190
1191 ;;; struktex.el ends here.
1192 ⟨/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.

[LaT20] The LaTeX3 Project. *The* l3build *package—Checking and building packages*, Januar 2020.

[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 Komödie*, 8(4):23–40, Februar 1996.

[Rah92] Sebastian Rahtz. *The* oz *package*, 1999.