

# struktex.sty\*

Jobst Hoffmann  
Fachhochschule Aachen, Abt. Jülich  
Ginsterweg 1  
52428 Jülich  
Bundesrepublik Deutschland

gedruckt am 4. Juni 2018

## Zusammenfassung

Dieser Artikel beschreibt den Einsatz und die Implementation der  $\LaTeX$ -*package* `struktex.sty` zum Setzen von Struktogrammen nach Nassi-Shneiderman.

## Inhaltsverzeichnis

|          |  |          |          |   |           |
|----------|--|----------|----------|---|-----------|
| <b>1</b> | <b>Lizenzvereinbarung</b>  | <b>1</b> | <b>5</b> | <b>Beispieldatei zum Einbinden in die Dokumentation</b>   | <b>23</b> |
| <b>2</b> | <b>Vorwort</b>   | <b>1</b> | <b>6</b> | <b>Verschiedene Beispieldateien</b>   | <b>24</b> |
| <b>3</b> | <b>Hinweise zur Pflege und Installation sowie die Treiberdatei zur Erzeugung dieser Dokumentation</b>        | <b>3</b> | 6.1      | Beispieldatei Nr. 1 . . . . .   | 24        |
| <b>4</b> | <b>Die Benutzungsschnittstelle</b>   | <b>5</b> | 6.2      | Beispieldatei Nr. 2 . . . . .   | 24        |
| 4.1      | Spezielle Zeichen und Textdarstellung . . . . .  | 6        | 6.3      | Beispieldatei Nr. 3 . . . . .   | 25        |
| 4.2      | Makros zur Darstellung von Variablen, Schlüsselwörtern und anderen programmierspezifischen Details . . . . . | 7        | 6.4      | Beispieldatei Nr. 4 . . . . .   | 30        |
| 4.3      | Die Makros zur Erzeugung von Struktogrammen . . . . .  | 9        | <b>7</b> | <b>Makros zur Erstellung der Dokumentation des <code>struktex.sty</code></b>  | <b>32</b> |
|          |  |          | <b>8</b> | <b>Makefile</b>   | <b>36</b> |
|          |  |          | <b>9</b> | <b>Stil Datei zur einfachen Eingabe von Struktogrammen beim Arbeiten mit dem (X)emacs und AUCT<math>\TeX</math></b> | <b>42</b> |

---

\*Diese Datei hat die Versionsnummer v2.3c-0-g7d3fc5b, wurde zuletzt bearbeitet am 2018/06/04, und die Dokumentation datiert vom 2018/06/04.

# 1 Lizenzvereinbarung

This package is copyright © 1995 – 2018 by:

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

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

# 2 Vorwort

St<sub>u</sub>kT<sub>E</sub>X hat eine lange Entwicklung hinter sich. Bei der Entwicklung wurden verschiedene Programme zur Versionsverwaltung – cvs, subversion und aktuell git – eingesetzt, die alle unterschiedliche Möglichkeiten bieten, Versionsnummern zu definieren. Um diese unterschiedlichen Versionsnummern zeitlich zuordnen zu können, wird folgendes Schema verwendet: Versionsnummern der Form „v- $\langle d \rangle$ . $\langle d \rangle$ [ $\langle a \rangle$ ]“ mit  $\langle d \rangle$  als Dezimalzahl und  $\langle a \rangle$  als Buchstabe, etwa v-4.1a bezeichnen die erste Entwicklungslinie (rcs). Die folgenden Versionsnummern haben die Form „v $\langle n \rangle$  $\langle n \rangle$  $\langle n \rangle$ “ mit  $\langle n \rangle$  als Dezimalziffer, z. B. v122 (subversion). Die aktuelle Entwicklung erfolgt unter git und benutzt Versionsnummern der Form „v( $\langle d \rangle$ . $\langle d \rangle$ )[ $\langle a \rangle$ ]- $\langle d \rangle$ -g $\langle x \rangle$ “, beispielsweise v2.1-13-gd28a927;  $\langle x \rangle$  ist dabei eine Hexadezimalzahl. Allgemein gilt für das Alter und somit die Reihenfolge der Versionen

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

Mit dem hier beschriebenen Makropaket ist es möglich, Struktogramme mit L<sup>A</sup>T<sub>E</sub>X zu zeichnen. Das Makropaket wird im folgenden immer St<sub>u</sub>kT<sub>E</sub>X genannt. Es ist in der Lage, die wichtigsten Elemente eines Struktogrammes wie z. B. Verarbeitungsblöcke, Schleifenkonstrukte, Sinnbilder für Alternativen usw. zu generieren. Die Struktogramme werden mit Hilfe der Picture-Umgebung von L<sup>A</sup>T<sub>E</sub>X erzeugt.<sup>1</sup>

Ab Version v-4.1a werden die mathematischen Symbole von A<sup>M</sup>S-T<sub>E</sub>X geladen, die den mathematischen Zeichensatz erweitern und andere Darstellungen von Mengensymbolen (etwa  $\mathbb{N}$ ,  $\mathbb{Z}$  und  $\mathbb{R}$  für die natürlichen, ganzen und reellen Zahlen) ermöglichen. Insbesondere das Zeichen für die leere Menge ( $\emptyset$ ) ist in der Darstellung auffälliger als das standardmäßige Zeichen („ $\emptyset$ “) und somit besser für die Darstellung von Struktogrammen geeignet.

Weiterhin ist aus dem `oz.sty` die Idee übernommen, Variablenamen in *italics* zu setzen, ohne dass die teilweise unschönen Zwischenräume erzeugt werden.

Die Entwicklung dieses Makropaketes ist noch nicht abgeschlossen. Es war geplant, die Struktogramme unter dem Einsatz des Makros aus `emlines2.sty` zu zeichnen, um die durch L<sup>A</sup>T<sub>E</sub>X gegebenen Einschränkungen – es gibt nur vordefinierte Steigungen – aufzuheben. Dies ist – für das `\ifthenelse` mit den Versionen v-4.1a und v-4.1b, für das `\switch` mit der Version v-4.2a – erledigt, nicht jedoch

---

<sup>1</sup> Wer es scheut, Struktogramme mittels L<sup>A</sup>T<sub>E</sub>X direkt zu schreiben, kann beispielsweise unter <http://structorizer.fisch.lu/> ein Programm (Strukturizer) finden, mit dem man seine Struktogramme mittels Maus entwickeln und abschließend als L<sup>A</sup>T<sub>E</sub>X-Code exportieren kann.

für Systeme, die die entsprechenden `\special{...}`-Befehle nicht unterstützen. Erreicht werden kann dies jedoch durch Einsatz entsprechender Makros aus dem `curves.sty`. Seit der Version v-8.0a wird das Paket `pict2e.sty` unterstützt, das mittels der üblichen Treiber die von der `picture`-Umgebung bekannten Beschränkungen auf nur wenige Steigungen im wesentlichen aufhebt, so dass sich die Benutzung der entsprechenden Option (s.u.) dauerhaft empfiehlt.

Ebenso ist es geplant, Struktogramme um Kommentarblöcke zu erweitern, wie sie in dem Buch von Futschek ([Fut89]) eingesetzt werden. Dieses ist ebenfalls mit der Version v-8.0a realisiert worden.

Weitere Zukunftspläne sind:

1. Ein `\otherwise`-Zweig beim `\switch` (abgeschlossen durch die Version v-4.2a).
2. Die Neuimplementation der `declaration`-Umgebung mittels der `list`-Umgebung gemäß [GMS94, Abs. 3.3.4] (abgeschlossen mit der Version v-4.5a).
3. Die Anpassung an  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$  im Sinne eines Packages (abgeschlossen durch die Version v-4.0a).
4. Die Verbesserung der Dokumentation, um Teile des Algorithmus verständlicher zu machen.
5. Die Unabhängigkeit des `struktex.sty` von anderen `.sty`-Dateien wie etwa dem `JHfMakro.sty` (abgeschlossen mit der Version v-4.5a).
6. Die vollständige Implementation der Makros `\pVar`, `\pKey`, `\pFonts`, `\pTrue`, `\pFalse` und `\pBoolValue` (erledigt vor Version v-7.0).
7. Die vollständige Internalisierung von Kommandos, die nur in der Umgebung `struktogramm` Sinn machen. Internalisierung bedeutet, dass diese Kommandos nur innerhalb der Umgebung definiert sind. Dies hat den Zweck, das Paket mit anderen Paketen verträglicher zu gestalten, etwa mit dem `ifthenelse.sty`. Begonnen wurde die Internalisierung mit der Version v-4.4a.
8. Die Unabhängigkeit der Dokumentation von anderen `.sty`-Dateien wie etwa dem `JHfMakro.sty` (abgeschlossen mit der Version v-5.0).
9. Eine alternative Darstellung der Deklarationen, wie sie von Rico Bolz vorgeschlagen wurde
10. Wiedereinführung der `make`-Ziele `dist-src`, `dist-tar` und `dist-zip`.

Der derzeitige Stand der Implementierung ist an entsprechender Stelle vermerkt.

### 3 Hinweise zur Pflege und Installation sowie die Treiberdatei zur Erzeugung dieser Dokumentation

Das Paket, zu dem der `struktex.sty` gehört, besteht aus insgesamt sechs Dateien:

```

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

```

Um daraus einerseits die Dokumentation, andererseits die .sty-Datei zu erzeugen, muss folgendermaßen vorgegangen werden:

Zunächst wird mit z. B.

```
tex struktex.ins
```

die Datei `struktex.ins` formatiert. Dieser Formatierungslauf erzeugt elf weitere Dateien. Dies sind zunächst die drei .sty-Dateien `struktex.sty`, `struktxf.sty` und `struktxp.sty`, die beim Einsatz des `struktex.sty` benötigt werden; weiterhin sind es die beiden Dateien `struktex.test_0.nss` und `strukdoc.sty`, die zur Erzeugung der hier vorliegenden Dokumentation benötigt werden. Dazu kommen drei Testdateien `struktex.test_i.nss`,  $i = 1(2)3$ , sowie die beiden Dateien `struktex.makemake` und `struktex.mk` (vgl. Abschnitt 8).

Die Dokumentation wird wie üblich durch

```

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

```

erzeugt.<sup>2</sup> Das Ergebnis dieses Formatierlaufes ist die Dokumentation in Form einer .pdf-Datei, die in gewohnter Weise weiterbearbeitet werden kann. Weitere Informationen zum Arbeiten mit der integrierten Dokumentation findet man in [Mit01] und [MDB01].

Die Installation wird abgeschlossen, indem die Datei `struktex.sty` in ein Verzeichnis verschoben wird, das von T<sub>E</sub>X gefunden werden kann, das ist in einer TDS-konformen Installation typischerweise `.../tex/latex/struktex/`, die Dokumentation wird analog in das Verzeichnis `.../doc/latex/struktex/` verschoben.<sup>3</sup>

Sollen Änderungen durchgeführt werden, so sollten neben diesen die Werte von `\fileversion`, `\filedate` und `\docdate` bei Bedarf entsprechend geändert werden. Weiterhin sollte darauf geachtet werden, dass die Revisionsgeschichte durch Einträge von

```
\changes{<Version>}{<Datum>}{<Kommentar>}
```

weitergeschrieben wird. `<Version>` gibt die Versionsnummer an, unter der die jeweilige Änderung durchgeführt wurde, `<Datum>` gibt das Datum in der Form `yy/mm/dd` an und `<Kommentar>` erläutert die jeweilige Änderung. `<Kommentar>` darf nicht mehr als 64 Zeichen enthalten. Daher sollten Kommandos nicht mit dem „\“ (*backslash*) eingeleitet werden, sondern mit dem „“ (*accent*).

<sup>2</sup>Die Erzeugung der Dokumentation kann durch den Einsatz einer `make`-Datei vereinfacht werden, vgl. Abschnitt 8

<sup>3</sup>Wenn die automatische Installation (vgl. Abschnitt 8) vorgenommen wird, erfolgt diese entsprechend.

Die nächsten Anweisungen bilden den Treiber für die hier vorliegende Dokumentation.

```

1 (*driver)
2                                     % select the formatting language:
3 \expandafter\ifx\csname primarylanguage\endcsname\relax%
4 \def\primarylanguage{ngerman}%
5 \def\secondarylanguage{english}%
6 \else%
7 \def\secondarylanguage{ngerman}%
8 \fi
9
10 \documentclass[a4paper, \secondarylanguage, % select the language
11     \primarylanguage]{ltxdoc}
12
13 \PassOptionsToPackage{obeyspaces}{url} % must be done before any package is
14                                     % loaded
15
16 \usepackage{babel} % for switching the documentation language
17 \usepackage{strukdoc} % the style-file for formatting this
18                                     % documentation
19
20 \usepackage[pict2e, % <----- to produce finer results
21                                     % visible under xdvi, alternatives are
22                                     % curves or emlines2 (visible only under
23                                     % ghostscript), leave out if not
24                                     % available
25     verification,
26     outer, % <----- to set the position of the \ifthenelse
27                                     % flags to the outer edges
28     debug,
29 ]
30 {struktex}
31 \GetFileInfo{struktex.sty}
32
33 \EnableCrossrefs
34 %\DisableCrossrefs % say \DisableCrossrefs if index is ready
35
36 %\RecordChanges % say \RecordChanges to gather update information
37
38 %\CodelineIndex % say \CodelineIndex to index entry code by line number
39
40 \OnlyDescription % say \OnlyDescription to omit the implementation details
41
42 \MakeShortVerb{\|} % |\foo| acts like \verb+\foo+
43
44 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
45 % to avoid underfull ... messages while formatting two/three columns
46 \hbadness=10000 \vbadness=10000
47
48 \typeout{\string\primarylanguage: \primarylanguage, \string\language: \the\language}
49
50 \def\languageNGerman{10} % depends on language.dat, put
51 % \the\language here
52

```

```

53 \begin{document}
54 \makeatletter
55 \ifundefined{selectlanguageEnglish}{\selectlanguage{english}}
56 \makeatother
57 \DocInput{struktex.dtx}
58 \end{document}
59 </driver>

```

## 4 Die Benutzungsschnittstelle

Der `struktex.sty` wird wie jede andere `.sty`-Datei als *package* in ein  $\text{\LaTeX}$ -Dokument eingebunden:

```
\usepackage[Optionen]{struktex}
```

Die folgenden Optionen stehen zur Verfügung:

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

Die jeweilige Option legt die Sprache für definierte Werte wie `\sTrue` fest, Standardwert ist `english`.

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

Durch Angabe einer der drei Optionen ist es möglich, beliebige Steigungen in Struktogrammen zu zeichnen. Erstere Option ist sinnvoll, wenn mit dem  $\text{emTeX}$ -Paket von Eberhard Mattes gearbeitet wird (DOS oder OS/2), ansonsten wird der Einsatz von `pict2e` empfohlen. Der Einsatz des Paketes `curves.sty` (Ian Maclaine-cross), das das Zeichnen von Geraden beliebiger Steigungen durch das Setzen vieler einzelner Punkte ermöglicht, ist durch das Erscheinen des Paketes `pict2e.sty` (Hubert Gäßlein und Rolf Niepraschk) im Prinzip obsolet, aus Kompatibilitätsgründen wird er weiter unterstützt. Durch die Angabe einer der genannten Option wird das jeweilige Paket (`emlines2.sty`, `curves.sty` bzw. `pict2e.sty`) automatisch geladen, Standardwert ist `pict2e`.

3. `verification`:

Nur wenn diese Option gesetzt ist, steht `\assert` als Kommando zur Verfügung.

4. `nofiller`:

Setzen dieser Option lässt jeden Freiraum leer. Ansonsten wird Freiraum in Alternativen als  $\emptyset$  markiert.

5. `draft`, `final`:

Diese Optionen dienen in üblicher Weise dazu, den Entwurf beziehungsweise die endgültige Fassung zu kennzeichnen (vgl. `\sProofOn`/`\sProofOff`). Im Entwurfsmodus werden die vier Eckpunkte eines Struktogramms in der vom Benutzer vorgegebenen Größe ausgegeben, diese Markierung fällt in der endgültigen Fassung weg. Der Standardwert ist `final`.

6. `debug`:

Mit dem Setzen dieser Option werden Zeilen der Form

==> dbg *<Text>*

in die .log-Datei geschrieben.

#### 7. outer:

Das Setzen dieser Option führt dazu, die ja/nein Flaggen statt auf der Mitte der Grundlinie jeweils links bzw. rechts außen erscheinen zu lassen.

Nach dem Laden der .sty-Datei stehen verschiedene Kommandos und Umgebungen zur Verfügung, die das Zeichnen der Struktogramme ermöglichen.

`\StrukTeX` Zunächst sei der Befehl erwähnt, der das Logo `StrukTeX` erzeugt:

`\StrukTeX`

Damit kann in Dokumentationen auf die hier vorliegende Stil-Option verwiesen werden.

## 4.1 Spezielle Zeichen und Textdarstellung

`\nat` Wegen ihres häufigen Auftretens sind die Mengen der natürlichen, ganzen, reellen  
`\integer` und komplexen Zahlen ( $\mathbb{N}$ ,  $\mathbb{Z}$ ,  $\mathbb{R}$  und  $\mathbb{C}$ ) im Mathematik-Modus über die folgenden  
`\real` Makros erreichbar: `\nat`, `\integer`, `\real` und `\complex`. Ebenso ist das mit  
`\complex` `\emptyset` erzeugte „ $\emptyset$ “ als Zeichen für die leere Anweisung auffälliger als das  
`\emptyset` standardmäßige Zeichen „ $\emptyset$ “. Andere Mengensymbole wie  $\mathbb{L}$  (für Lösungsmenge)  
sind über `\mathbb{L}` zu erzeugen.

`\MathItalics` Mit diesen beiden Makros kann die Darstellung von Variablenamen beeinflusst  
`\MathNormal` werden:

*NeuerWert = AlterWert + Korrektur* `\MathNormal`  
`\[`  
`NeuerWert = AlterWert + Korrektur`  
`\]`

und

*NeuerWert = AlterWert + Korrektur* `\MathItalics`  
`\[`  
`NeuerWert = AlterWert + Korrektur`  
`\]`

## 4.2 Makros zur Darstellung von Variablen, Schlüsselwörtern und anderen programmierspezifischen Details

`\pVariable` Struktogramme enthalten manchmal direkt zu programmierenden Code. Um  
`\pVar` hier ein einheitliches Aussehen zu erreichen, sind die hier aufgeführten Makros  
`\pKeyword` definiert worden. Um diese Makros auch in anderem Zusammenhang nutzen zu  
`\pKey` können, sind sie zu einem eigenen *package* `strukt.sty` zusammengefasst worden.  
`\pComment` Dabei wird ab Version 122 zur Darstellung von Code auf das Paket „`url.sty`“ von  
Donald Arsenau zurückgegriffen, das es ermöglicht, verbatim gesetzte Texte als  
Parameter an ein anderes Makro zu übergeben. Wenn diese Texte ein Leerzeichen

enthalten, das erhalten bleiben soll, muss der Benutzer vor dem Laden von `url.sty`, typischerweise also vor der Anweisung

```
\usepackage{struktex}
```

die Anweisung

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

setzen.

Mit `\pVariable{<Variablenname>}` wird ein Variablenname gesetzt. `<Variablenname>` ist dabei ein Bezeichner eine Variable, wobei der Unterstrich „\_“, das kaufmännische Und „&“ und das Dach „^“ als Teile des Variablennamens erlaubt sind:

|  |  |
|--|--|
| <code>cEineNormaleVariable</code>            | <code>\obeylines</code>                                  |
| <code>c_eine_normale_Variable</code>         | <code>\renewcommand{\pLanguage}{C}</code>                |
| <code>&amp;iAdresseEinerVariablen</code>     | <code>\pVariable{cEineNormaleVariable}</code>            |
| <code>zZeigerAufEineVariable^.sInhalt</code> | <code>\pVariable{c_eine_normale_Variable}</code>         |
|  | <code>\pVariable{&amp;iAdresseEinerVariablen}</code>     |
|  | <code>\renewcommand{\pLanguage}{Pascal}</code>           |
|  | <code>\pVariable{zZeigerAufEineVariable^.sInhalt}</code> |

Leerzeichen werden beachtet, so dass ganze Anweisungen geschrieben werden können. Es darf als Abkürzung `\pVar` benutzt werden.

Entsprechend wird mit `\pKeyword{<Schlüsselwort>}` ein Schlüsselwort gesetzt. Dabei ist `<Schlüsselwort>` ein Schlüsselwort in einer Programmiersprache, wobei der Unterstrich „\_“ und das *hash*-Zeichen „#“ als Teil des Schlüsselwortes erlaubt ist. Damit lässt sich setzen:

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

Auch `\pKeyword` darf abgekürzt werden: `\pKey`. Damit erzeugt dann der Quelltext

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

als Ausgabe dieses Ergebnis:

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

In ähnlicher Weise dient `\pComment` zur Darstellung von Kommentar. Das Argument darf nur Zeichen der *TeX*-Kategorie *letter* haben, Zeichen, die einen Kommentar einleiten, müssen geschrieben werden. `\pComment` kann nicht abgekürzt werden. Beispielsweise ergibt

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

die Zeile

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

`\pTrue` Logische Werte spielen in der Programmierung eine wesentliche Rolle. Mit  
`\pFalse` `\pTrue` und `\pFalse` sind entsprechende Werte vorgegeben: WAHR und FALSCH.  
`\pFonts` Der Makro `\pFonts` dient der Auswahl von Fonts zur Darstellung von Varia-  
`\pBoolValue` blen, Schlüsselwörtern und Kommentar:

```
\pFonts{<Variablenfont>}{<Schlüsselwortfont>} {<Kommentarfont>}
```

Vorbesetzt sind die einzelnen Fonts mit

- `<Variablenfont>` als `\small\sffamily`,
- `<Schlüsselwortfont>` als `\small\sffamily\bfseries` und
- `<Kommentarfont>` als `\small\sffamily\slshape`.

Damit wird die obige Zeile nach

```
\pFonts{\itshape}{\sffamily\bfseries}{\scshape}  
\pVar{a = }\pKey{sqrt}\pVar{a);} \pComment{// Iteration}
```

zu

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

Entsprechend können durch den Makro

```
\sBoolValue{<Ja-Wert>}{<Nein-Wert>}
```

die Werte von `\pTrue` und `\pFalse` undefiniert werden. Somit liefern die Zeilen

```
\renewcommand{\pLanguage}{Pascal}  
\sBoolValue{\textit{ja}}{\textit{nein}}  
\pFalse{} = \pKey{not} \pTrue
```

das folgende Ergebnis:

```
nein = not ja
```

`\sVar` Die Makros `\sVar` und `\sKey` sind mit den Makros `\pVar` und `\pKey` iden-  
`\sKey` tisch, sie werden hier nur definiert, um Kompatibilität mit früheren Versionen  
`\sTrue` des `struktex.sty` zu gewährleisten. Dasselbe gilt auch für die Makros `\sTrue` und  
`\sFalse` `\sFalse`.

### 4.3 Die Makros zur Erzeugung von Struktogrammen

`struktogramm` Die Umgebung

```
\sProofOn  
\sProofOff  
\PositionNSS  
  
\begin{struktogramm}(<Breite>,<Höhe>) [<Überschrift>]  
...  
\end{struktogramm}
```

erzeugt Raum für ein neues Struktogramm. Die beiden Parameter legen die Breite und die Höhe des Platzes fest, der für das Struktogramm reserviert wird. Die Angaben werden in Millimetern gemacht, wobei der aktuelle Wert von `\unitlength` keine Rolle spielt. Dabei entspricht die Breite der tatsächlichen Breite, die tatsächliche Höhe wird den Erfordernissen angepasst. Stimmt die angegebene Höhe nicht mit den tatsächlichen Erfordernissen überein, läuft das Struktogramm in den umgebenden Text hinein oder es bleibt Raum frei. Es gibt einen Schalter `\sProofOn`, mit dem die angegebene Größe des Struktogramms durch vier Punkte gezeigt wird, um Korrekturen einfacher durchführen zu können. `\sProofOff` schaltet diese Hilfe entsprechend wieder ab. Die Überschrift dient zur Identifizierung des Struktogramms, wenn man sich von anderer Stelle, etwa aus einem anderen Struktogramm heraus auf dieses hier beziehen will.

Die Struktogramm-Umgebung basiert auf der `picture`-Umgebung von  $\text{\LaTeX}$ . Die in der `picture`-Umgebung gebräuchliche Längeneinheit `\unitlength` wird bei den Struktogrammen nicht benutzt; die Längeneinheit ist aus technischen Gründen auf  $1\text{ mm}$  festgelegt. Weiterhin müssen alle Längenangaben ganzzahlige Werte sein. `\unitlength` hat zwar nach dem Zeichnen eines Struktogramms mit  $\text{\TeX}$  den gleichen Wert wie vorher, ist aber innerhalb eines Struktogramms undefiniert und darf dort auch nicht geändert werden.

`\assign` Das Hauptelement eines Struktogramms ist ein Kasten, in dem eine Operation beschrieben wird. Ein solcher Kasten wird mit `\assign` erzeugt. Die Syntax ist

$$\backslash\text{assign}[\langle\text{Höhe}\rangle]\{\langle\text{Inhalt}\rangle\},$$

wobei die eckigen Klammern wie üblich ein optionales Argument bezeichnen. Die Breite und die Höhe des Kastens werden den Erfordernissen gemäß automatisch angepasst, man kann aber mittels des optionalen Argumentes die Höhe des Kastens vorgeben.

Der *Text* wird normalerweise linksbündig in den Kasten gesetzt; ist er dafür zu lang, so wird ein Paragraph (im Blocksatz) gesetzt.

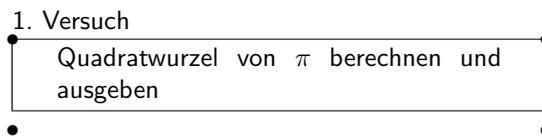
### Beispiel 1

Ein einfaches Struktogramm wird mit den folgenden Anweisungen erzeugt:

```
\sProofOn
\begin{struktogramm}(70,12)[1.\ Versuch]
  \assign{Quadratwurzel von  $\pi$  berechnen und ausgeben}
\end{struktogramm}
\sProofOff
```

Diese Anweisungen führen zu folgendem Struktogramm, wobei der Anwender wie auch bei der zugrundeliegenden `picture`-Umgebung für eine geeignete Positionierung zu sorgen hat. Die Positionierung erfolgt in dieser Dokumentation im Regelfall mit der `quote`-Umgebung, man kann ein Struktogramm aber auch mit der `center`-Umgebung zentrieren. Die Breite des Struktogramms ist mit  $70\text{mm}$  vorgegeben, die Höhe mit  $12\text{mm}$ . Eine Alternative ist durch die `centernss`-Umgebung gegeben, die auf Seite 21 beschrieben wird.

Gleichzeitig wird die Wirkung von `\sProofOn` und `\sProffOff` gezeigt, wobei die zu große vorgegebene Größe des Struktogramms zu beachten ist.



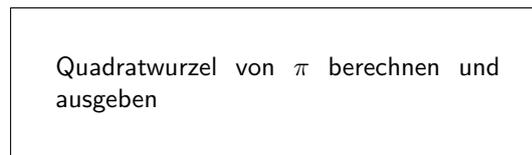
Die Bedeutung des optionalen Argumentes macht das folgende Beispiel deutlich.

### Beispiel 2

Die Höhe des Kastens wird vorgegeben:

```
\begin{center}
\begin{struktogramm}(70,20)
  \assign[20]{Quadratwurzel von $\pi$ berechnen und ausgeben}
\end{struktogramm}
\end{center}
```

Diese Anweisungen führen zu folgendem Struktogramm, wobei zu beachten ist, dass die `struktogramm`-Umgebung mittels einer `center`-Umgebung zentriert wurde, wobei die Breite des Struktogramms wiederum mit 70mm vorgegeben ist, die Höhe diesmal aber mit 20mm.



**declaration** Die `declaration`-Umgebung dient der Beschreibung von Variablen bzw. der Beschreibung der Schnittstelle. Ihre Syntax ist

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

**\declarationtitle** Die Überschriftsangabe ist optional. Lässt man die Angabe weg, so wird standardmäßig die Überschrift: „Speicher bereitstellen.“ erzeugt. Will man einen anderen Text haben, wird dieser mit `\declarationtitle{\langle Überschrift \rangle}` global festgelegt. Will man für ein einzelnes Struktogramm einen speziellen Titel erzeugen, so gibt man diesen in den eckigen Klammern an.

**\description** Innerhalb der `declaration`-Umgebung werden die Beschreibungen der einzelnen Variablen mit

```
\descriptionwidth
\descriptionsep
\description{\langle Variablenname \rangle}{\langle Variablenbeschreibung \rangle}
```

erzeugt. Dabei ist zu beachten, dass `\langle Variablenname \rangle` keine schließende eckige Klammer „]“ beinhalten darf, da dieser Makro mittels des `\item`-Makros definiert worden ist. Eckige Klammern sind in diesem Fall als `\lbracket` und `\rbracket` einzugeben.

Das Aussehen einer Beschreibung lässt sich mit drei Parametern steuern: `\descriptionindent`, `\descriptionwidth` und `\descriptionsep`; die Bedeutung der Parameter ist der Abbildung 1 zu entnehmen (`\xsize@nss` und `\xin@nss` sind interne Größen, die von `StuTeX` vorgegeben werden). Die Vorbesetzung dieser Werte ist folgendermaßen:

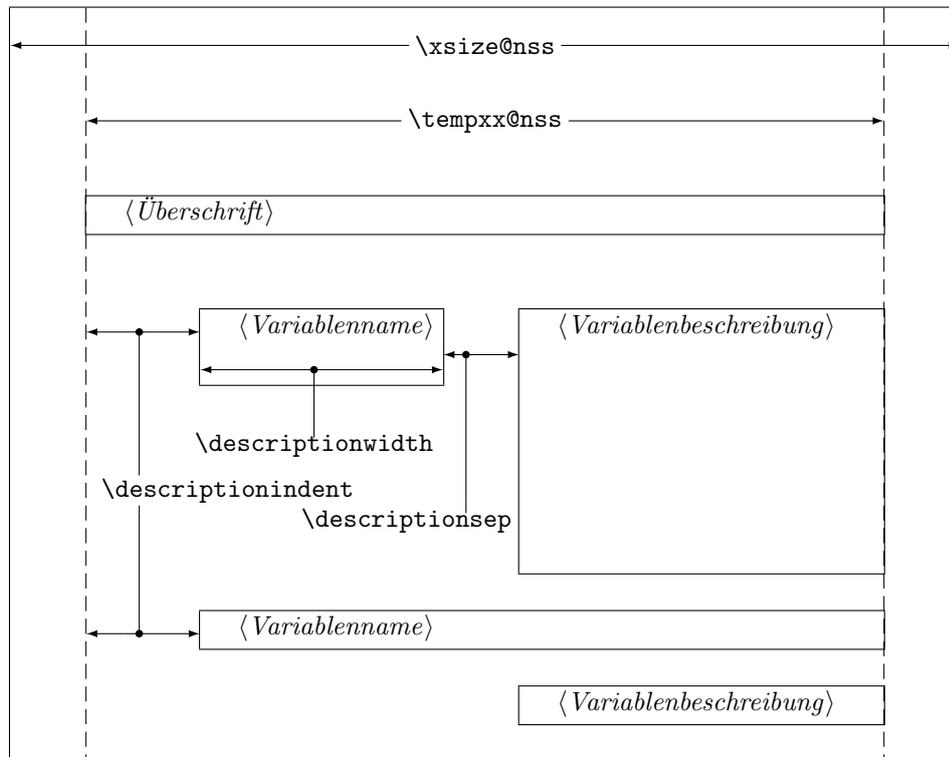


Abbildung 1: Aufbau einer Variablenbeschreibung

```

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

```

Die Bedeutung von `\descriptionwidth` ist darin zu sehen, dass ein Variablenname, der kürzer als `\descriptionwidth` ist, eine Beschreibung erhält, die auf der gleichen Höhe liegt; ansonsten wird die Beschreibung eine Zeile tiefer begonnen.

### Beispiel 3

Zunächst wird nur eine einzelne Variable beschrieben.

```

\begin{struktogramm}(95,20)
  \assign%
  {%
    \begin{declaration}
      \description{\pVar{iVar}}{eine \pKey{int}-Variable,
        deren Beschreibung hier allein dem
        Zweck dient, den Makro vorzuf"uhren}
    \end{declaration}
  }
\end{struktogramm}

```

Das zugehörige Struktogramm, wobei zu beachten ist, dass durch die leeren eckigen Klammern keine Überschrift erzeugt wird.

Speicherplatz bereitstellen:

```
iVar {eine int -Variable, deren Beschreibung hier allein dem Zweck dient, den Makro vorzuführen}
```

Nun werden Variablen genauer spezifiziert:

```
\begin{struktogramm}(95,50)
  \assign%
    \begin{declaration}[Parameter:]
      \description{\pVar{iPar}}{ein \pKey{int}-Parameter,
        dessen Bedeutung hier beschrieben wird}
    \end{declaration}
    \begin{declaration}[lokale Variablen:]
      \description{\pVar{iVar}}{eine \pKey{int}-Variable,
        deren Bedeutung hier beschrieben wird}
      \description{\pVar{dVar}}{eine \pKey{double}-Variable,
        deren Bedeutung hier beschrieben wird}
    \end{declaration}
  }
\end{struktogramm}
```

Das ergibt:

```
Parameter:
  iVar {ein int -Parameter, dessen Bedeutung
        hier beschrieben wird}
lokale Variablen:
  iVar {eine int -Variable, deren Bedeutung
        hier beschrieben wird}
  dVar {eine double -Variable, deren Bedeutung
        hier beschrieben wird}
```

Zuletzt die globale Vereinbarung eines Titels:

```
\def\declarationtitle{globale Variablen:}
\begin{struktogramm}(95,13)
  \assign%
    \begin{declaration}
      \description{\pVar{iVar_g}}{eine \pKey{int}-Variable}
    \end{declaration}
  }
\end{struktogramm}
```

Dies ergibt das folgende Aussehen:

```
globale Variablen:
  iVar_g {eine int -Variable}
```

Hier ist die lokale Umsetzung des `\catcodes` des Unterstrichs zu beachten, die erforderlich ist, wenn man einen Unterstrich in einem Makroargument einsetzen möchte. Diese lokale Umsetzung wird zwar auch schon bei `\pVar` gemacht, reicht aber bei der Makroexpansionstechnik von  $\text{T}\text{E}\text{X}$  nicht aus.

`\sub` Die Sinnbilder für einen Unterprogrammsprung und einen Aussprung aus dem  
`\return` Programm sehen ähnlich aus und werden mit folgenden Befehlen gezeichnet:

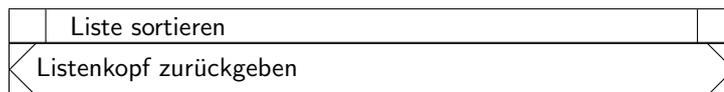
```
\sub[⟨Höhe⟩]{⟨Text⟩}
\return[⟨Höhe⟩]{⟨Text⟩}
```

Die Parameter haben dieselbe Bedeutung wie bei `\assign`. Das nächste Beispiel zeigt, wie diese Sinnbilder gezeichnet werden.

#### Beispiel 4

```
\begin{struktogramm}(95,20)
  \sub{Liste sortieren}
  \return{Listenkopf zur"uckgeben}
\end{struktogramm}
```

Diese Anweisungen führen zu folgendem Struktogramm:



`\while` Zum Darstellen von Schleifenkonstrukten stehen die drei Befehle `\while`,  
`\whileend` `\until` und `\forever` zur Verfügung. Die While-Schleife stellt eine Wiederholung  
`\until` mit vorausgehender Bedingungsprüfung (kopfgesteuerte Schleife) dar, die Until-  
`\untilend` Schleife testet die Bedingung am Schleifenende (fußgesteuerte Schleife) und die  
`\forallin` Forever-Schleife ist eine Endlosschleife, aus der man mit dem Befehl `\exit` her-  
`\forallinend` ausspringen kann.

```
\forever      \while[⟨Breite⟩]{⟨Text⟩}⟨Unterstruktogramm⟩\whileend
\foreverend  \until[⟨Breite⟩]{⟨Text⟩}⟨Unterstruktogramm⟩\untilend
              \forever[⟨Breite⟩]⟨Unterstruktogramm⟩\foreverend
              \exit[⟨Höhe⟩]{⟨Text⟩}
```

`⟨Breite⟩` ist die Dicke des Rahmens des Sinnbildes, `⟨Text⟩` ist der Bedingungstext, der in diesen Rahmen geschrieben wird. Wird die Breite nicht angegeben, richtet sich die Rahmendicke nach der Höhe des Textes. Der Text wird linksbündig in den Rahmen geschrieben. Ist der Text leer, so wird ein dünner Rahmen gezeichnet.

Ein Kontrollkonstrukt, das heute in vielen Programmiersprachen verfügbar ist, ist eine Schleife, die abhängig von der jeweiligen Sprache als `forall`-, `for ... in` oder `foreach`-Schleife bezeichnet wird. Diese Schleifenart kann prinzipiell als kopfgesteuerte Schleife angesehen werden, manche ziehen aber eine Schreibweise vor, die von der Endlosschleife abgeleitet wird. Für diesen Fall gibt es das Konstrukt

```
\forallin[⟨Breite⟩]{⟨Text⟩}⟨Unterstruktogramm⟩\forallinend
```

An Stelle von  $\langle \text{Unterstruktogramm} \rangle$  können beliebige Befehle von  $\text{St}_{\text{u}}\text{T}_{\text{E}}\text{X}$  stehen (mit Ausnahme von  $\backslash\text{openstrukt}$  und  $\backslash\text{closestrukt}$ ), die das Struktogramm innerhalb der  $\backslash\text{while}$ -, der  $\backslash\text{until}$ - oder der  $\backslash\text{forever}$ -Schleife bilden.

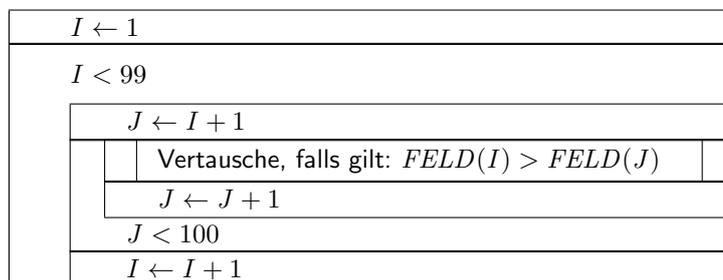
Um Kompatibilität mit der Weiterentwicklung des  $\text{struktex.sty}$  von J. Dietel zu erreichen, gibt es die Makros  $\backslash\text{dfr}$  und  $\backslash\text{dfrend}$  mit derselben Bedeutung wie  $\backslash\text{forever}$  und  $\backslash\text{foreverend}$ .

Die beiden folgenden Beispiele zeigen den Einsatz der  $\backslash\text{while}$ - und  $\backslash\text{until}$ -Makros sowie der  $\backslash\text{forallin}$ -Makros,  $\backslash\text{forever}$  wird weiter unten gezeigt.

### Beispiel 5

```
\begin{struktogramm}(95,40)
  \assign{(I \gets 1)}
  \while[8]{(I < 99)}
    \assign{(J \gets I+1)}
    \until{(J < 100)}
      \sub{Vertausche, falls gilt: (FELD(I) > FELD(J))}
      \assign{(J \gets J+1)}
    \untilend
  \assign{(I \gets I+1)}
\whileend
\end{struktogramm}
```

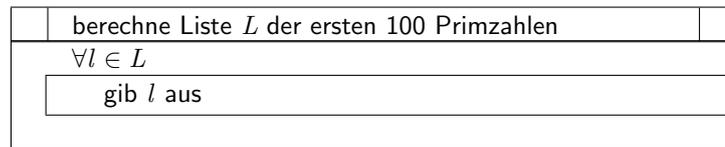
Diese Anweisungen führen zu folgendem Struktogramm:



### Beispiel 6

```
\begin{struktogramm}(95, 25)
  \sub{berechne Liste (L) der ersten 100 Primzahlen}
  \forallin{(\forall l \in L)}
    \assign{gib (l) aus}
  \forallinend
\end{struktogramm}
```

Diese Anweisungen führen zu folgendem Struktogramm:



Die `\exit`-Anweisung gibt nur im Zusammenhang mit einfachen oder mehrfachen Verzweigungen Sinn, daher wird sie im Anschluss an die Diskussion der Verzweigungen vorgestellt.

`\ifthenelse` Zur Darstellung von Alternativen stellt `StukTeX` die Sinnbilder für einen If-Then-Else-Block und für mehrfache Alternativen eine Case-Konstruktion zur Verfügung. Da in der klassischen `picture`-Umgebung von `LATEX` nur Linien mit bestimmten Steigungen gezeichnet werden können, muss der Benutzer bei beiden Befehlen selbst den Winkel bestimmen, mit dem die notwendigen schrägen Linien gezeichnet werden sollen (hier ist also etwas mehr ‚Handarbeit‘ nötig).

Wenn hingegen der `curves.sty`, der `emlines2.sty` oder der `pict2e.sty` eingesetzt wird, ist die Darstellung von Geraden mit beliebiger Steigung möglich.

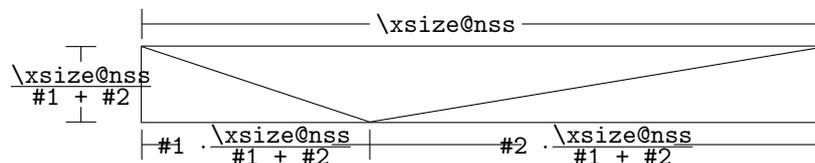
Der If-Then-Else-Befehl sieht so aus:

```

\ifthenelse[ $\langle$ Höhe $\rangle$ ]{ $\langle$ Linker Winkel $\rangle$ }{ $\langle$ Rechter Winkel $\rangle$ }
  { $\langle$ Bedingung $\rangle$ }{ $\langle$ Linker Text $\rangle$ }{ $\langle$ Rechter Text $\rangle$ }
   $\langle$ Unterstruktogramm $\rangle$ 
\change
   $\langle$ Unterstruktogramm $\rangle$ 
\ifend

```

Für den Fall, dass das optionale Argument  $\langle$ Höhe $\rangle$  nicht angegeben ist, sind  $\langle$ Linker Winkel $\rangle$  (#1) und  $\langle$ Rechter Winkel $\rangle$  (#2) Ziffern zwischen 1 und 6; diese bestimmen die Steigung der beiden Unterteilungslinien des If-Then-Else-Blocks (großer Wert = kleine Steigung). Größere Werte werden auf 6 gesetzt, kleinere auf 1. Das genaue Verhalten der Steigungen ist dem folgenden Bild zu entnehmen; `\xsize@nss` ist dabei die Breite des aktuellen Unterstruktogrammes. Wird die  $\langle$ Höhe $\rangle$  vorgegeben, so bestimmt dieser Wert statt des Ausdruckes  $\frac{\text{\xsize@nss}}{\#1 + \#2}$  die Höhe des Bedingungsrechteckes.



$\langle$ Bedingung $\rangle$  wird in das so gebildete obere mittlere Dreieck gesetzt; die Parameter  $\langle$ Linker Text $\rangle$  und  $\langle$ Rechter Text $\rangle$  werden in das linke bzw. rechte untere Dreieck gesetzt. Der Bedingungs-Text kann in seinem Dreiecks-Feld umgebrochen werden. Ab Version v-5.3 wird der Bedingungs-Text durch geeigneten Umbruch beliebigen Steigungen angepasst.<sup>4</sup> Die beiden anderen Texte sollten kurz sein (z. B. ja/nein oder true/false), da sie nicht umgebrochen werden können und sonst über ihr

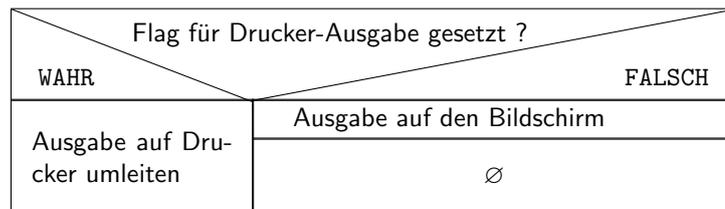
<sup>4</sup>Diese Erweiterung stammt von Daniel Hagedorn, dem ich hiermit herzlich danken möchte

Dreiecks-Feld hinausragen. Um an dieser Stelle Einheitlichkeit zu erzielen, sollten die Makros `\pTrue` und `\pFalse` benutzt werden. Hinter `\ifthenelse` werden die Befehle für das linke, hinter `\change` die für das rechte „Unterstruktogramm“ geschrieben. Falls diese beiden Struktogramme nicht gleich lang sind, wird ein Kasten mit einem  $\emptyset$  ergänzt.<sup>5</sup> Mit `\ifend` wird das If-Then-Else-Element beendet. Es folgen zwei Beispiele für die Anwendung.

### Beispiel 7

```
\begin{struktogramm}(95,32)
  \ifthenelse[12]{1}{2}
    {Flag f"ur Drucker-Ausgabe gesetzt ?}{\sTrue}{\sFalse}
    \assign[15]{Ausgabe auf Drucker umleiten}
  \change
    \assign{Ausgabe auf den Bildschirm}
  \ifend
\end{struktogramm}
```

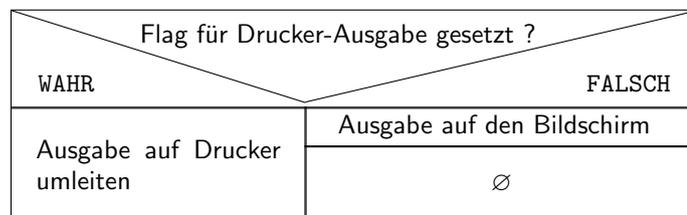
Diese Anweisungen führen zu folgendem Struktogramm:



### Beispiel 8

```
\begin{struktogramm}(90,30)
  \ifthenelse{3}{4}
    {Flag f"ur Drucker-Ausgabe gesetzt ?}{\sTrue}{\sFalse}
    \assign[15]{Ausgabe auf Drucker umleiten}
  \change
    \assign{Ausgabe auf den Bildschirm}
  \ifend
\end{struktogramm}
```

Diese Anweisungen führen zu folgendem Struktogramm:



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

Das Case-Konstrukt hat folgende Syntax:

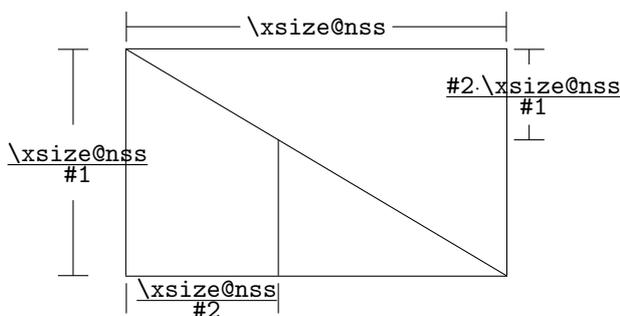
<sup>5</sup>Eventuell ist ein `\strut` hilfreich, um unterschiedliche Höhen von Texten auszugleichen.

```

\case[⟨Höhe⟩]{⟨Winkel⟩}{⟨Anzahl der Fälle⟩}{⟨Bedingung⟩}{⟨Text
des 1. Falles⟩
  ⟨Unterstruktogramm⟩
\switch[⟨Position⟩]{⟨Text des 2. Falles⟩
  ⟨Unterstruktogramm⟩
...
\switch[⟨Position⟩]{⟨Text des n. Falles⟩
  ⟨Unterstruktogramm⟩
\caseend

```

Ist die  $\langle \text{Höhe} \rangle$  nicht angegeben, so erhält die Unterteilungslinie des Case-Sinnbildes die durch  $\langle \text{Winkel} \rangle$  angegebene Steigung (die bei `\ifthenelse` erwähnten Winkelwerte). In das obere der durch diese Linie entstandenen beiden Dreieck wird der Text  $\langle \text{Bedingung} \rangle$  gesetzt. Die Größenverhältnisse ergeben sich aus der folgenden Skizze ( $\backslash\text{size@nss}$  ist die aktuelle Breite des (Unter-)Struktogramms):



Der zweite Parameter  $\langle \text{Anzahl der Fälle} \rangle$  gibt die Anzahl der zu zeichnenden Fälle an; alle Unterstruktogramme der einzelnen Fälle erhalten die gleiche Breite. Der  $\langle \text{Text des 1. Falles} \rangle$  muss als Parameter des `\case`-Befehls angegeben werden, alle weiteren Fälle werden über den `\switch`-Befehl eingeleitet. Hinter dem Text folgen dann die Befehle für das eigentliche Unterstruktogramm des jeweiligen Falles. Der letzte Fall wird mit `\caseend` abgeschlossen. Ein Case-Sinnbild mit drei Fällen zeigt das folgende Beispiel.

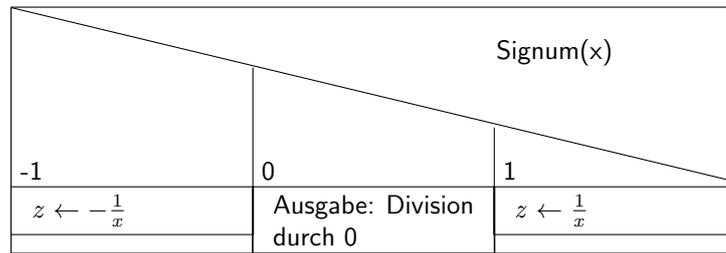
### Beispiel 9

```

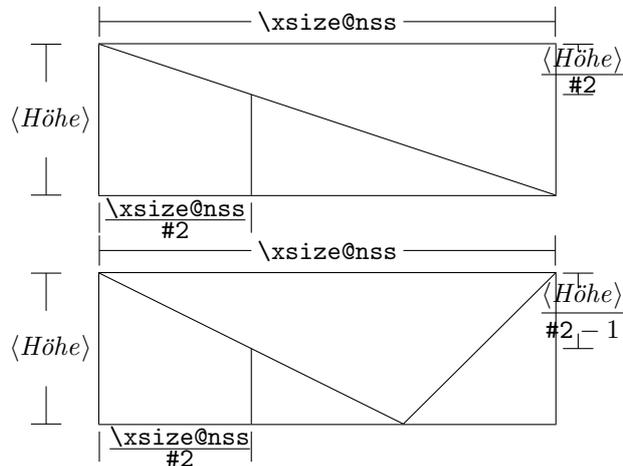
\begin{struktogramm}(95,30)
  \case{4}{3}{Signum(x)}{-1}
    \assign{$z \gets - \frac{1}{x}$}
  \switch{0}
    \assign{Ausgabe: Division durch 0}
  \switch{1}
    \assign{$z \gets \frac{1}{x}$}
  \caseend
\end{struktogramm}

```

Diese Anweisungen führen zu folgendem Struktogramm:



Der optionale Parameter [ $\langle H\ddot{o}he \rangle$ ] ist nur einzusetzen, wenn die Option „curves“, „emlines2“ oder „pict2e“ gesetzt ist; ist das nicht der Fall, können die Struktogramme durcheinander kommen. Die Erweiterung des `\switch`-Kommandos mit [ $\langle H\ddot{o}he \rangle$ ] führt zu einer anderen Bedeutung von  $\langle Winkel \rangle$ . Ist der Wert gerade, wird wie zuvor eine gerade Linie zur Aufteilung des zugrundeliegenden Rechtecks gezeichnet; ist der Wert hingegen ungerade, wird der letzte Fall wie im folgenden gezeigt als Sonderfall gezeichnet.



### Beispiel 10

```

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

```

Diese Anweisungen führen zu folgendem Struktogramm:

| Signum(x)                   |                           |                            |
|-----------------------------|---------------------------|----------------------------|
| -1                          | 0                         | 1                          |
| $z \leftarrow -\frac{1}{x}$ | Ausgabe: Division durch 0 | $z \leftarrow \frac{1}{x}$ |

Ist der erste Parameter hingegen ungerade, wird ein Standardzweig gezeichnet; der Wert für den Standardfall sollte dann rechtsbündig ausgerichtet werden.

#### Beispiel 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[r]{0}
    \assign{Ausgabe: Division durch 0}
  \caseend
\end{struktogramm}
```

Diese Anweisungen führen zu folgendem Struktogramm:

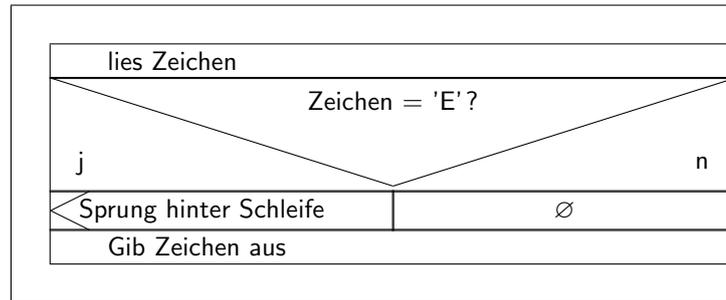
| Signum(x)                   |                            |                           |
|-----------------------------|----------------------------|---------------------------|
| -1                          | 1                          | 0                         |
| $z \leftarrow -\frac{1}{x}$ | $z \leftarrow \frac{1}{x}$ | Ausgabe: Division durch 0 |

Das folgende Beispiel zeigt, wie mittels einfacher Verzweigung aus einer Endlosschleife gesprungen werden kann. Das Beispiel lässt sich ohne weiteres auf eine mehrfache Verzweigung übertragen.

#### Beispiel 12

```
\begin{struktogramm}(95,40)
  \forever
    \assign{lies Zeichen}
    \ifthenelse{3}{3}{Zeichen = 'E'?}
      {j}{n}
      \exit{Sprung hinter Schleife}
    \change
  \ifend
  \assign{Gib Zeichen aus}
\foreverend
\end{struktogramm}
```

Diese Anweisungen führen zu folgendem Struktogramm:

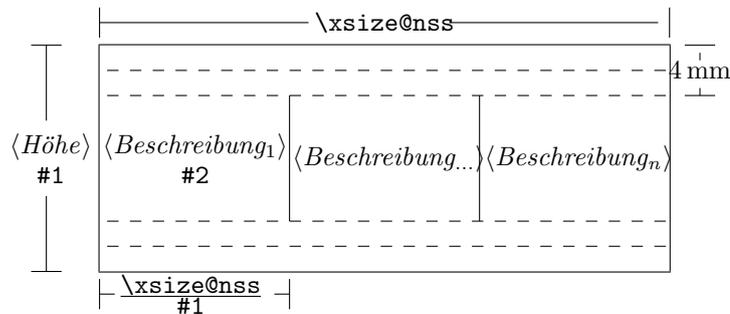


`\inparallel`      Heutzutage sind Prozessoren mit mehreren Kernen oder auch massive Parallelrechner ein übliches Werkzeug zur Ausführung von Programmen. Um die Fähigkeiten dieser Prozessoren auszunutzen, sollten entsprechende parallele Algorithmen entwickelt und implementiert werden. Der Makro `\inparallel` und die zugehörigen Makros `\task` und `\inparalleleend` ermöglichen die Darstellung paralleler Verarbeitung in einem Programm. Die Syntax lautet:

```

\inparallel[<Höhe der 1. Task>】{<Anzahl paralleler
Tasks>】{<Beschreibung der 1. Task>】}
\task[<position>】{<Beschreibung der 2. Task>】}
...
\task[<position>】{<Beschreibung der n. Task>】}
\inparalleleend
  
```

Das Layout eines mit diesen Kommandos erzeugten Kastens ist der folgenden Abbildung zu entnehmen (die Makroparameter #1 und #2 beziehen sich auf die Parameter von `\inparallel`):



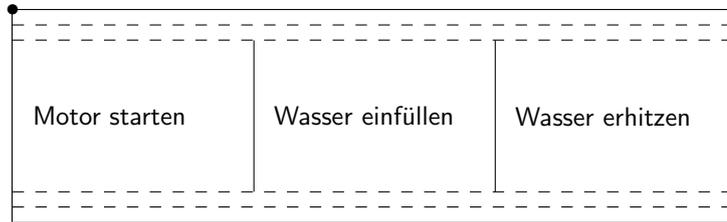
Zu beachten ist, dass die verschiedenen Tasks nicht weiter gegliedert werden dürfen.

**Beispiel 13** (Anwendung von `\inparallel`)

```

\begin{struktogramm}(95,40)
  \inparallel[20]{3}{Motor starten}
  \task{Wasser einf"ullen}
  \task{Wasser erhitzen}
  \inparalleleend
\end{struktogramm}
  
```

Diese Anweisungen ergeben das folgende Struktogramm:



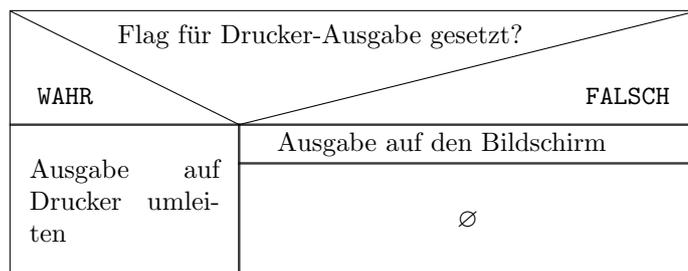
`centernss` Soll ein Struktogramm zentriert dargestellt werden, so wird dazu die Umgebung

```
\begin{centernss}
  <Struktogramm>
\end{centernss}
```

benutzt:

```
\begin{centernss}
\begin{struktogramm}(90,35)
  \ifthenelse{2}{4}
    {Flag f"ur Drucker-Ausgabe gesetzt?}{\sTrue}{\sFalse}%
    \assign[20]{Ausgabe auf Drucker umleiten}
  \change
    \assign{Ausgabe auf den Bildschirm}
  \ifend
\end{struktogramm}
\end{centernss}
```

Das führt zu folgendem:



`\CenterNssFile` Häufig gibt es den Fall, dass Struktogramme in eigenen Dateien abgelegt werden, damit sie für sich allein auf Korrektheit getestet werden können oder in anderen Zusammenhängen genutzt werden können. Sollen sie zentriert eingebunden werden, kann nicht mit der folgenden Konstruktion gearbeitet werden:

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

da auf diese Weise der gesamte Text innerhalb des Struktogramms zentriert würde. Um diesen Fall einfach und korrekt abhandeln zu können, kann das Makro `\CenterNssFile` eingesetzt werden, das auch in der Schreibweise `centernssfile` definiert ist. Voraussetzung ist, dass die Datei, die die Anweisungen für das Struktogramm enthält, die Dateinamenserweiterung `.nss` hat, der Name der einzubindenden Datei *muß* demzufolge ohne Erweiterung angegeben werden. Wenn die Datei `struktex-test-0.nss` das in Abschnitt 5, Zeile 2–10 gezeigte Aussehen hat, so führt die Anweisung

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

zu folgendem Aussehen des formatierten Textes:

|                             |                                |                            |
|-----------------------------|--------------------------------|----------------------------|
| Text                        |                                |                            |
|                             |                                | Signum(x)                  |
| -1                          | 0                              | 1                          |
| $z \leftarrow -\frac{1}{x}$ | Ausgabe: Divisi-<br>on durch 0 | $z \leftarrow \frac{1}{x}$ |

`\openstrukt` Diese beiden Makros sind nur der Kompatibilität zu vorherigen Versionen  
`\closestrukt` von  $\text{\LaTeX}$  willen noch erhalten. Von der Bedeutung her entsprechen sie `\struktogramm` und `\endstruktogramm`. Die Syntax ist

```
\openstrukt{\width }{\height }
```

und

```
\closestrukt.
```

`\assert` Der Makro `\assert` wurde eingeführt, um die Verifikation von Algorithmen zu unterstützen, er ist aber nur aktiv, wenn die Stil-Option `verification` gesetzt wurde. Er dient dazu, an ausgewählten Stellen Zusicherungen über den Zustand von Variablen zu markieren, die Syntax entspricht dem `\assign`:

```
\assert [Höhe] {Zusicherung},
```

Sein Einsatz ergibt sich aus dem folgenden:

```
\begin{struktogramm}(70,20)[Zusicherungen in Struktogrammen]
  \assign{\(a\gets a^2\)}
  \assert{\(a\ge 0\)}
\end{struktogramm}
```

Das dazugehörige Struktogramm sieht folgendermaßen aus:

|                                 |
|---------------------------------|
| Zusicherungen in Struktogrammen |
| $a \leftarrow a^2$              |
| $a \geq 0$                      |

## 5 Beispieldatei zum Einbinden in die Dokumentation

Die folgenden Zeilen bilden eine Beispieldatei, die bei der Erstellung dieser Dokumentation benötigt wird.

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

## 6 Verschiedene Beispieldateien

### 6.1 Beispieldatei zum Austesten der Makros des `struktex.sty` ohne die Benutzung optionaler Pakete

Die folgenden Zeilen bilden eine Musterdatei, die zum Austesten der Makros benutzt werden kann. Der Inhalt ist nur in Englisch vorhanden.

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

```

99   \assign{(a\gets b)}
100  \assign{(b\gets tmp)}
101  \ifend
102 \end{struktogramm}
103
104 \end{document}
105 \end{example2}

```

## 6.2 Beispieldatei zum Austesten der Makros des `struktex.sty` mit dem Paket `pict2e.sty`

Die folgenden Zeilen bilden eine Musterdatei, die zum Austesten der Makros benutzt werden kann. Der Inhalt ist nur in Englisch vorhanden.

```

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

```

```

146     \assign[\StruktBoxHeight]{\(\a\gets b\)}
147     \assign[\StruktBoxHeight]{\(\b\gets tmp\)}
148     \assert[\StruktBoxHeight]{\(\a<b\le c\)}
149   \ifend
150   \assert[\StruktBoxHeight]{\(\a\le b \le c\)}
151 \end{struktoqramm}
152
153 \end{document}
154 </example3>

```

### 6.3 Beispieldatei zum Austesten der Makros des `struktxp.sty`

Die folgenden Zeilen bilden eine Musterdatei, die zum Austesten der Makros des `struktxp.sty` benutzt werden kann. Zum Testen sollten auch die Kommentarzeichen vor der Zeile `\usepackage[T1]{fontenc}` gelöscht werden. Der Text ist nur in Englisch vorgegeben.

```

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

```

```

192 }
193
194 \paragraph{after changing the fonts with}
195 \verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:
196
197 {\obeylines
198 \pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
199 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
200 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
201 in math mode: \(\pVar{a}+\pVar{iV_g}\)
202 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
203 }
204
205 \paragraph{after changing the fonts with}
206 \verb-\pFonts{\itshape}{\bfseries\itshape}{}-:
207
208 {\obeylines
209 \pFonts{\itshape}{\bfseries\itshape}{}
210 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
211 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
212 in math mode: \(\pVar{a}+\pVar{iV_g}\)
213 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
214
215 \vspace{15pt}
216 Without \textit{italic correction}:
217     M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
218 }
219
220 \pLanguage{C}
221 \pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
222 \section*{Default values (C):}
223
224 {\obeylines
225 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
226 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
227 in math mode: \(\pVar{a}+\pVar{iV_g}\)
228 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
229 }
230
231 \paragraph{After changing the boolean values with}
232 \verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:
233
234 {\obeylines
235 \pBoolValue{\texttt{yes}}{\texttt{no}}
236 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
237 }
238
239 \paragraph{after changing the fonts with}
240 \verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:
241
242 {\obeylines
243 \pFonts{\itshape}{\sffamily\bfseries}{}
244 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
245 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}

```

```

246 in math mode: \(\pVar{a}+\pVar{iV_g}\)
247 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
248 }
249
250 \paragraph{after changing the fonts with}
251 \verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:
252
253 {\obeylines
254 \pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
255 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
256 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
257 in math mode: \(\pVar{a}+\pVar{iV_g}\)
258 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
259 }
260
261 \paragraph{after changing the fonts with}
262 \verb-\pFonts{\itshape}{\bfseries\itshape}{}-:
263
264 {\obeylines
265 \pFonts{\itshape}{\bfseries\itshape}{}
266 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
267 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
268 in math mode: \(\pVar{a}+\pVar{iV_g}\)
269 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
270
271 \vspace{15pt}
272 Without \textit{italic correction}:
273     M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
274 }
275
276 \pLanguage{Java}
277 \pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
278 \section*{Default values (Java):}
279
280 {\obeylines
281 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
282 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
283 in math mode: \(\pVar{a}+\pVar{iV_g}\)
284 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
285 }
286
287 \paragraph{After changing the boolean values with}
288 \verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:
289
290 {\obeylines
291 \pBoolValue{\texttt{yes}}{\texttt{no}}
292 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
293 }
294
295 \paragraph{after changing the fonts with}
296 \verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:
297
298 {\obeylines
299 \pFonts{\itshape}{\sffamily\bfseries}{}

```

```

300 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
301 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
302 in math mode: \(\pVar{a}+\pVar{iV_g}\)
303 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
304 }
305
306 \paragraph{after changing the fonts with}
307 \verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:
308
309 {\obeylines
310 \pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
311 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
312 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
313 in math mode: \(\pVar{a}+\pVar{iV_g}\)
314 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
315 }
316
317 \paragraph{after changing the fonts with}
318 \verb-\pFonts{\itshape}{\bfseries\itshape}{}-:
319
320 {\obeylines
321 \pFonts{\itshape}{\bfseries\itshape}{}
322 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
323 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
324 in math mode: \(\pVar{a}+\pVar{iV_g}\)
325 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
326
327 \vspace{15pt}
328 Without \textit{italic correction}:
329 M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
330 }
331
332 \pLanguage{Python}
333 \pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
334 \section*{Default values (Python):}
335
336 {\obeylines
337 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
338 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
339 in math mode: \(\pVar{a}+\pVar{iV_g}\)
340 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
341 }
342
343 \paragraph{After changing the boolean values with}
344 \verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:
345
346 {\obeylines
347 \pBoolValue{\texttt{yes}}{\texttt{no}}
348 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
349 }
350
351 \paragraph{after changing the fonts with}
352 \verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:
353

```

```

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

```

## 6.4 Beispieldatei zum Austesten der Makros des `strukt xp.sty`

Die folgenden Zeilen werden in einem anderen Zusammenhang benutzt, Java-Methoden zu dokumentieren. An dieser Stelle wird ein eigener Weg gewählt, da das sonst übliche Arbeiten mit `lstinl ine` zu Fehlern führt.

```

392 (*example5)
393 \documentclass{article}
394
395 \usepackage{strukt xp, strukt xf}
396
397 \makeatletter
398 \newlength{\fdesc@len}
399 \newcommand{\fdesc@label}[1]%
400 {%

```

```

401 \settowidth{\fdesc@len}{\fdesc@font #1}%
402 \advance\hsize by -2em
403 \ifdim\fdesc@len>\hsize% % term > labelwidth
404 \parbox[b]{\hsize}%
405 {%
406 \fdesc@font #1%
407 }\\%
408 \else% % term < labelwidth
409 \ifdim\fdesc@len>\labelwidth% % term > labelwidth
410 \parbox[b]{\labelwidth}%
411 {%
412 \makebox[0pt][l]{\fdesc@font #1}\\%
413 }%
414 \else% % term < labelwidth
415 {\fdesc@font #1}%
416 \fi\fi%
417 \hfil\relax%
418 }
419 \newenvironment{fdescription}[1][\tt]%
420 {%
421 \def\fdesc@font{#1}
422 \begin{quote}%
423 \begin{list}{}%
424 {%
425 \renewcommand{\makelabel}{\fdesc@label}%
426 \setlength{\labelwidth}{120pt}%
427 \setlength{\leftmargin}{\labelwidth}%
428 \addtolength{\leftmargin}{\labelsep}%
429 }%
430 }%
431 {%
432 \end{list}%
433 \end{quote}%
434 }
435 \makeatother
436
437 \pLanguage{Java}
438
439 \begin{document}
440
441 \begin{fdescription}
442 \item[\index{Methoden}>drawImage(Image img,
443 int dx1,
444 int dy1,
445 int dx2,
446 int dy2,
447 int sx1,
448 int sy1,
449 int sx2,
450 int sy2,
451 ImageObserver observer)=%
452 \Expr{\pKey{public} \pKey{abstract} \pKey{boolean} drawImage(Image img,
453 \pKey{int} dx1,
454 \pKey{int} dy1,

```

```

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

```

## 7 Makros zur Erstellung der Dokumentation des `struktex.sty`

Um die Formatierung der Dokumentation ein wenig zu erleichtern, werden ein paar Makros benutzt, die in einer eigenen `.sty`-Datei zusammengefasst wurden. Ein wesentlicher Teil beruht auf einer Modifikation der `newtheorem`-Umgebung aus `latex.sty` zur Auszeichnung der Beispiele, die Implementation der Abkürzungen wurde in [Neu96] vorgeschlagen.

Dazu wurden aus dem `verbatim.sty` einige Kommandos übernommen und modifiziert, damit das Schreiben und Lesen von Dateien im `verbatim`-Modus auch im Zusammenhang mit dem `docstrip`-Paket funktioniert. Schließlich wurde auch noch eine Idee von Tobias Oetiker aus seinem `layout.sty`, der im Zusammenhang mit *lshort2e.tex - The not so short introduction to LaTeX2e* entstand, zum parallelen Setzen von Quelle und formatiertem Text genutzt.

```

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

```

```

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

```

```

545 \verbatim@start}%
546 {\immediate\closeout\struktex@out\@esphack\endgroup%
547 %
548 % And here comes the part of Tobias Oetiker
549 %
550 \par\small\addvspace{3ex plus 1ex}\vskip -\parskip
551 \noindent
552 \makebox[0.45\linewidth][l]{%
553 \begin{minipage}[t]{0.45\linewidth}
554 \vspace*{-2ex}
555 \setlength{\parindent}{0pt}
556 \setlength{\parskip}{1ex plus 0.4ex minus 0.2ex}
557 \begin{trivlist}
558 \item\input{\jobname.tmp}
559 \end{trivlist}
560 \end{minipage}}%
561 \hfill%
562 \makebox[0.5\linewidth][l]{%
563 \begin{minipage}[t]{0.50\linewidth}
564 \vspace*{-1ex}
565 \verbatiminput{\jobname.tmp}
566 \end{minipage}}
567 \par\addvspace{3ex plus 1ex}\vskip -\parskip
568 }
569
570 \newtoks\verbatim@line
571 \def\verbatim@startline{\verbatim@line{}}
572 \def\verbatim@addtoline#1{%
573 \verbatim@line\expandafter{\the\verbatim@line#1}}
574 \def\verbatim@processline{\the\verbatim@line\par}
575 \def\verbatim@finish{\ifcat$\the\verbatim@line$\else
576 \verbatim@processline\fi}
577
578 \def\verbatimwrite#1{%
579 \@bsphack
580 \immediate\openout \struktex@out #1
581 \let\do\@makeother\dospecials
582 \catcode'\^^M\active \catcode'\^^I=12
583 \def\verbatim@processline{%
584 \immediate\write\struktex@out
585 {\the\verbatim@line}}%
586 \verbatim@start}
587 \def\endverbatimwrite{%
588 \immediate\closeout\struktex@out
589 \@esphack}
590
591 \@ifundefined{vrb@catcodes}%
592 {\def\vrb@catcodes{%
593 \catcode'\!12\catcode'\[12\catcode'\]12}}{}
594 \begingroup
595 \vrb@catcodes
596 \lccode'\!='\ \lccode'\[='\ \lccode'\]='\
597 \catcode'\~=\active \lccode'\~='\^^M
598 \lccode'\C='\C

```

```

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

```

```

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

```

```
707 {\typeout {No file #2.}\endgroup}}
708 </strukdoc>
```

## 8 Makefile zur automatisierten Erstellung der Dokumentation und der Tests des `struktex.sty`

Der Umgang mit `.dtx`-Paketen ist wesentlich einfacher, wenn ein Werkzeug für die Automatisierung der wesentlichen Schritte vorliegt. Für Unix/Linux basierte Systeme ist das mit `make` und einem geeigneten `Makefile` einfach zu realisieren. Hier wird der `Makefile` in die Dokumentation integriert, die spezielle Syntax mit Tabulatorzeichen wird durch ein Hilfsprogramm erzeugt, das weiter unten angegeben ist. Auf die Benutzung des `Makefile` wird hier nicht weiter eingegangen, da der erfahrene Benutzer des Werkzeugs diese aus der Datei selbst entnehmen kann.

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

```

748 #INSTALLTEXMF='kpsewhich --expand-var '$$TEXMFLOCAL'
749 # But you may set INSTALLTEXMF to every directory you want.
750 # Use following, if you only want to test the installation:
751 #INSTALLTEXMF=/tmp/texmf
752
753 # If texhash must run after installation, you can invoke this:
754 TEXHASH=texhash
755
756 ##### Edit following only, if you want to change defaults!
757
758 # The directory, where to install *.cls and *.sty
759 CLSDIR=$(INSTALLTEXMF)/tex/latex/$(PACKAGE)
760
761 # The directory, where to install documentation
762 DOCDIR=$(INSTALLTEXMF)/doc/latex/$(PACKAGE)
763
764 # The directory, where to install the sources
765 SRCDIR=$(INSTALLTEXMF)/source/latex/$(PACKAGE)
766
767 # The directory, where to install demo-files
768 # If we have some, we have to add following 2 lines to install rule:
769 # $(MKDIR) $(DEMODIR); \
770 # $(INSTALL) $(DEMO_FILES) $(DEMODIR); \
771 DEMODIR=$(DOCDIR)/demo
772
773 # We need this, because the documentation needs the classes and packages
774 # It's not really a good solution, but it's a working solution.
775 TEXINPUTS := $(PWD):$(TEXINPUTS)
776
777 #####
778 # End of customization section
779 #####
780
781 LATEX = latex
782 PDFLATEX = pdflatex
783 TEX = TEX
784
785 COMMON_OPTIONS = # \OnlyDescription\CodelineNumbered\PageIndex
786 HISTORY_OPTIONS = \RecordChanges
787 DEVELOPER_OPTIONS = \EnableCrossrefs\RecordChanges\AlsoImplementation\CodelineIndex
788
789 # tarring options
790 EXgit = --exclude .git --exclude .gitignore --exclude auto --exclude tests \
791 --exclude *.tgz --exclude *.bib
792
793 # The name of the game
794 PACKAGE = struktex
795
796 DISTRIBUTION_FILES = ../$(PACKAGE)/$(PACKAGE).de.pdf \
797 ../$(PACKAGE)/$(PACKAGE).en.pdf \
798 ../$(PACKAGE)/$(PACKAGE).dtx \
799 ../$(PACKAGE)/$(PACKAGE).ins \
800 ../$(PACKAGE)/LIESMICH.md \
801 ../$(PACKAGE)/README.md

```

```

802 PACKAGE_FILES_A = $(subst ../$(PACKAGE)/,,$(DISTRIBUTION_FILES))
803 PACKAGE_FILES_B = $(subst $(PACKAGE).dtx ,,$(PACKAGE_FILES_A))
804 PACKAGE_FILES_C = $(subst $(PACKAGE).ins ,,$(PACKAGE_FILES_B))
805 PACKAGE_FILES_D = $(subst LIESMICH.md,,$(PACKAGE_FILES_C))
806 PACKAGE_FILES = $(subst README.md,,$(PACKAGE_FILES_D))
807
808 # To generate the version number of the distribution from the source
809 VERSION_L := git describe --long | xargs git --no-pager show -s \
810     --date=short --format=format:"$(PACKAGE) version ??? of %ad%n" |\
811     sed -e "s/????/'git describe --long'/"
812 VERSION_S := 'git describe --long | \
813     sed 's+-g.+++''
814
815 # to create the correct tar-file
816 define TAR_COMMANDS
817 echo $$@
818 OUT_DIR=$(mktemp -d)
819 mkdir $$OUT_DIR/struktex
820 cp $$@ $$OUT_DIR/struktex
821 pushd $$OUT_DIR
822 tar cfvz struktex.tgz struktex
823 popd
824 cp $$OUT_DIR/struktex.tgz .
825 endef
826
827 export TAR_COMMANDS
828
829 ## Main Targets
830
831 # strip off the comments from the package
832 $(PACKAGE).sty $(PACKAGE)-test-*.tex: $(PACKAGE).ins $(PACKAGE).dtx $(PACKAGE).sed
833 +$(TEX) $<; \
834 source $(PACKAGE).makemake; \
835 make revision_no; \
836 source $(PACKAGE).sed # set version number
837
838 all-de: $(PACKAGE).de.pdf
839
840 all-en: $(PACKAGE).en.pdf
841
842 # prepare the file with git revision information
843 .PHONY: revision_no
844 revision_no: $(PACKAGE).sed
845
846 $(PACKAGE).sed: $(PACKAGE).dtx
847 printf "%b\n" "set_git_info() {" \
848     > $(PACKAGE).sed; \
849 printf "%b\n" "sed -i -e 's/^[ \\t]*%% git revision information$$/\" \
850     >> $(PACKAGE).sed; \
851 git describe --long | \
852     xargs git --no-pager show -s --format=format:\
853     "\\@git@ \\\$Date: %ci $$$%\n" >> $(PACKAGE).sed; \
854 git describe --long | cut -c 2- | \
855     sed -e "s/~/\ \\\$Revision: /" -e "s/$$/ $$$\\\" \

```

```

856     >> $(PACKAGE).sed; \
857 git describe --long | \
858     xargs git --no-pager show -s --format=format:"    %%% \\\$Author: %an $$$\\n" \
859     >> $(PACKAGE).sed; \
860 printf "%b\n" "/" \\\$1" \
861     >> $(PACKAGE).sed; \
862 printf "%b\n" ";" \
863     >> $(PACKAGE).sed; \
864 printf "%b\n" "for f in \\\\" \
865     >> $(PACKAGE).sed; \
866 printf "%b\n" "\$(PACKAGE).sty \\\\" \
867     >> $(PACKAGE).sed; \
868 printf "%b\n" "struktxf.sty \\\\" \
869     >> $(PACKAGE).sed; \
870 printf "%b\n" "strukt xp.sty \\\\" \
871     >> $(PACKAGE).sed; \
872 printf "%b\n" "strukdoc.sty \\\\" \
873     >> $(PACKAGE).sed; \
874 printf "%b\n" "; do \\\\" \
875     >> $(PACKAGE).sed; \
876 printf "%b\n" " set_git_info \\\$f; done" \
877     >> $(PACKAGE).sed; \
878
879 # generate the documentation
880 $(PACKAGE).de.pdf: $(PACKAGE).dtx $(PACKAGE).sty struktex.sed
881 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{<}\"
882 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{<}\"
883 +mv $(<:.dtx=.pdf) $(<:.dtx=.de.pdf)
884
885 $(PACKAGE).en.pdf: $(PACKAGE).dtx $(PACKAGE).sty
886 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\primarylanguage{english}\input{<}\"
887 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\primarylanguage{english}\input{<}\"
888 +mv $(<:.dtx=.pdf) $(<:.dtx=.en.pdf)
889
890 # generate the documentation with revision history (only german)
891 history: $(PACKAGE).dtx $(PACKAGE).sty
892 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(HISTORY_OPTIONS)}\input{<}\"
893 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(HISTORY_OPTIONS)}\input{<}\"
894 +makeindex -s gind.ist $(PACKAGE).idx
895 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
896 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(HISTORY_OPTIONS)}\input{<}\"
897
898 # generate the documentation for the developer (revision history always
899 # in german)
900 develop-de: $(PACKAGE).dtx $(PACKAGE).sty
901 +$(PDFLATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)}$(DEVELOPER_OPTIONS)}\input{<}\"
902 +$(PDFLATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)}$(DEVELOPER_OPTIONS)}\input{<}\"
903 +makeindex -s gind.ist $(PACKAGE).idx
904 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
905 +$(PDFLATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)}$(DEVELOPER_OPTIONS)}\input{<}\"
906 +mv $(<:.dtx=.pdf) $(<:.dtx=.de.pdf)
907
908 develop-en: $(PACKAGE).dtx $(PACKAGE).sty
909 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(DEVELOPER_OPTIONS)}\def\primarylanguage{eng

```

```

910 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(DEVELOPER_OPTIONS)}\def\primarylanguage{eng
911 +makeindex -s gind.ist $(PACKAGE).idx
912 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
913 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(DEVELOPER_OPTIONS)}\def\primarylanguage{eng
914 +mv $(<:.dtx=.pdf) $(<:.dtx=.en.pdf)
915
916 # format the example/test files
917 test:
918 for i in `seq 1 3`; do \
919     f=$(PACKAGE)-test-$$i; \
920     echo file: $$f; \
921     $(PDFLATEX) $$f; \
922 done
923
924 install: $(PACKAGE).dtx $(PACKAGE).dvi
925 [ -d $(CLSDIR) ] || mkdir -p $(CLSDIR)
926 [ -d $(DOCDIR) ] || mkdir -p $(DOCDIR)
927 [ -d $(SRCDIR) ] || mkdir -p $(SRCDIR)
928 cp $(PACKAGE).sty $(CLSDIR)
929 cp $(PACKAGE).dvi $(DOCDIR)
930 cp $(PACKAGE).ins $(SRCDIR)
931 cp $(PACKAGE).dtx $(SRCDIR)
932 cp $(PACKAGE)-test-*.tex $(SRCDIR)
933 cp LIESMICH $(SRCDIR)
934 cp README $(SRCDIR)
935 cp THIS-IS-VERSION-$(VERSION) $(SRCDIR)
936
937 uninstall:
938 rm -f $(CLSDIR)/$(PACKAGE).sty
939 rm -fr $(DOCDIR)
940 rm -fr $(SRCDIR)
941
942 dist: $(PACKAGE).de.pdf $(PACKAGE).en.pdf $(PACKAGE).dtx $(PACKAGE).ins \
943 LIESMICH.md README.md
944 + echo "$$TAR_COMMANDS" > ./tar_commands; \
945 rm -f THIS_IS_VERSION_*; \
946 $(VERSION_L) > THIS_IS_VERSION_$(VERSION_S); \
947 sh ./tar_commands $^; \
948 mv ./struktex.tgz ./struktex-$(VERSION_S).tgz
949 rm ./tar_commands
950
951 tds-zip: $(PACKAGE_FILES)
952 + rm -f THIS_IS_VERSION_* *.zip; \
953 $(VERSION_L) | sed -e "s/?????????/$(VERSION_S)/" > THIS_IS_VERSION_$(VERSION_S); \
954 DOC_FILES="LIESMICH.md README.md THIS_IS_* $(PACKAGE).???.pdf"; \
955 MAKE_FILES="$(PACKAGE).m*"; \
956 SRC_FILES="$(PACKAGE).dtx $(PACKAGE).ins"; \
957 STY_FILES="struk*.sty"; \
958 TEST_FILES="./$(PACKAGE)-test*"; \
959 SUPPORT_FILES="./$(PACKAGE).el"; \
960 if [[ -d /tmp/texmf ]]; then \
961     rm -rf /tmp/texmf; \
962 fi; \
963 if [[ -f $(PACKAGE)-TDS.zip ]]; then \

```

```

964 rm $(PACKAGE)-TDS.zip; \
965 fi; \
966 mkdir -p /tmp/texmf/doc/latex/$(PACKAGE); \
967 mkdir -p /tmp/texmf/source/latex/$(PACKAGE); \
968 mkdir -p /tmp/texmf/tex/latex/$(PACKAGE); \
969 cp -a $$${DOC_FILES} /tmp/texmf/doc/latex/$(PACKAGE); \
970 cp -a $$${MAKE_FILES} /tmp/texmf/doc/latex/$(PACKAGE); \
971 cp -a $$${SRC_FILES} /tmp/texmf/source/latex/$(PACKAGE); \
972 cp -a $$${STY_FILES} /tmp/texmf/tex/latex/$(PACKAGE); \
973 cp -a $$${TEST_FILES} /tmp/texmf/doc/latex/$(PACKAGE); \
974 cp -a $$${SUPPORT_FILES} /tmp/texmf/doc/latex/$(PACKAGE); \
975 VERSION_SHORT="xxx"; \
976 pushd /tmp/texmf; \
977 zip -r /tmp/$(PACKAGE)-TDS.zip .; \
978 popd; \
979 mv /tmp/$(PACKAGE)-TDS.zip ./$(PACKAGE)-TDS-$(VERSION_S).zip
980
981
982 clean:
983 -rm -f *.log *.aux *.brf *.idx *.ilg *.ind
984 -rm -f *.glg *.glo *.gls *.lof *.lot *.out *.toc *.tmp *~
985 -rm *.mk *.makemake
986
987 realclean: clean
988 -rm -f *.sty *.cls *.pdf
989 -rm -f *-test-* Makefile
990
991 clean-test:
992 rm $(PACKAGE)-test-*. * # this $-sign is needed for font-locking in XEmacs only
993 </makefile>

```

Die folgende Zeile, die nach `latex` `struktex.ins` als Datei `struktex.makemake` vorliegt, kann mit dem Kommando

```
sh struktex.makemake
```

dazu benutzt werden, die obige Datei in ein Format umzusetzen, das von üblichen `make`-Programmen wie dem GNU `make` verarbeitet werden kann.

```

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

```

## 9 Stil Datei zur einfachen Eingabe von Struktogrammen beim Arbeiten mit dem (X)emacs und AUCTEX

Der (X)emacs und das Paket AUCTEX (<http://www.gnu.org/software/auctex/>) bilden ein mächtiges Werkzeug beim Erstellen von TEX/L<sup>A</sup>TEX-Dateien. Wenn es eine passende Stildatei für ein L<sup>A</sup>TEX-Paket wie das hier vorliegende St<sup>ü</sup>k<sub>k</sub>TEX gibt, wird die Eingabe von Quelltext durch automatisiertes Vervollständigen und viele andere Hilfsmittel sehr erleichtert. Im folgenden wird eine derartige Stildatei

bereitgestellt; sie muss nach ihrer Erzeugung noch an eine entsprechende Stelle kopiert werden.

Diese Datei ist zum jetzigen Zeitpunkt noch in der Entwicklungsphase, d. h. man kann mit ihr arbeiten, aber es fehlen noch ein paar Dinge wie das *font locking*, die Anzahl der automatisch eingefügten `\switch` Kommandos sollte nicht fest gleich eins sein, sondern von der Anzahl der eingegebenen Fälle abhängig sein.

```
997 (*auctex)
998 ;;; struktex.el --- AUCTeX style for 'struktex.sty'
999
1000 ;; Copyright (C) 2006 - 2018 Free Software Foundation, Inc.
1001
1002 ;; Author: J. Hoffmann <j.hoffmann_(at)_fh-aachen.de>
1003 ;; Maintainer: j.hoffmann_(at)_fh-aachen.de
1004 ;; Created: 2017/06/06
1005 ;; Keywords: tex
1006
1007 ;;; Commentary:
1008 ;; This file adds support for 'struktex.sty'
1009
1010 ;;; Code:
1011
1012 (TeX-add-style-hook
1013  "struktex"
1014  (lambda ()
1015    ;; Add declaration to the list of environments which have
1016    ;; an optional argument for each item.
1017    (add-to-list 'LaTeX-item-list
1018      '("declaration" . LaTeX-item-argument))
1019    (LaTeX-add-environments
1020     "centernss"
1021     '("struktogramm" LaTeX-env-struktogramm)
1022     '("declaration" LaTeX-env-declaration))
1023    (TeX-add-symbols
1024     '("PositionNSS" 1)
1025     '("assert" [ "Height" ] "Assertion")
1026     '("assign" [ "Height" ] "Statement")
1027     "StrukTeX"
1028     '("case" TeX-mac-case)
1029     "switch" "Condition"
1030     "caseend"
1031     '("declarationtitle" "Title")
1032     '("description" "Name" "Meaning")
1033     "emptyset"
1034     '("exit" [ "Height" ] "What" )
1035     '("forever" TeX-mac-forever)
1036     "foreverend"
1037     '("forallin" TeX-mac-forallin)
1038     "forallin"
1039     '("ifthenelse" TeX-mac-ifthenelse)
1040     "change"
1041     "ifend"
1042     '("inparallel" TeX-mac-inparallel)
1043     '("task" "Description"))
```

```

1044 "inparallelend"
1045 "sProofOn"
1046 "sProofOff"
1047 '("until" TeX-mac-until)
1048 "untilend"
1049 '("while" TeX-mac-while)
1050 "whileend"
1051 '("return" [ "Height" ] "Return value")
1052 '("sub" [ "Height" ] "Task")
1053 '("CenterNssFile" TeX-arg-file)
1054 '("centernssfile" TeX-arg-file))
1055 (TeX-run-style-hooks
1056 "pict2e"
1057 "emlines2"
1058 "curves"
1059 "struktxp"
1060 "struktxf"
1061 "ifthen")
1062 ;; Filling
1063 ;; Fontification
1064 ))
1065
1066 (defun LaTeX-env-struktogramm (environment)
1067 "Insert ENVIRONMENT with width, height specifications and
1068 optional title."
1069 (let ((width (read-string "Width: "))
1070       (height (read-string "Height: "))
1071       (title (read-string "Title (optional): ")))
1072 (LaTeX-insert-environment environment
1073                            (concat
1074                             (format "(%s,%s)" width height)
1075                             (if (not (zerop (length title)))
1076                                 (format "[%s]" title))))))
1077
1078 (defun LaTeX-env-declaration (environment)
1079 "Insert ENVIRONMENT with an optional title."
1080 (let ((title (read-string "Title (optional): ")))
1081 (LaTeX-insert-environment environment
1082                            (if (not (zerop (length title)))
1083                                (format "[%s]" title))))))
1084
1085 (defun TeX-mac-case (macro)
1086 "Insert \\case with all arguments, the needed \\switch(es) and
1087 the final \\caseend. These are optional height and the required
1088 arguments slope, number of cases, condition, and the texts for
1089 the different cases"
1090 (let ((height (read-string "Height (optional): "))
1091       (slope (read-string "Slope: "))
1092       (number (read-string "Number of cases: "))
1093       (condition (read-string "Condition: "))
1094       (text (read-string "Case no. 1: "))
1095       (count 1)
1096       )
1097 (setq number-int (string-to-number number))

```

```

1098 (insert (concat (if (not (zerop (length height)))
1099 (format "[%s]" height))
1100 (format "{%s}{%s}{%s}{%s}"
1101 slope number condition text)))
1102 (while (< count number-int)
1103 (end-of-line)
1104 (newline-and-indent)
1105 (newline-and-indent)
1106 (setq prompt (format "Case no. %d: " (+ 1 count)))
1107 (insert (format "\\switch{%s}" (read-string prompt)))
1108 (setq count (1+ count)))
1109 (end-of-line)
1110 (newline-and-indent)
1111 (newline-and-indent)
1112 (insert "\\caseend")))
1113
1114 (defun TeX-mac-forallin (macro)
1115 "Insert \\forallin-block with all arguments.
1116 This is the optional height and the description of the loop"
1117 (let ((height (read-string "Height (optional): "))
1118 (loop-description (read-string "Description of loop: ")))
1119 (insert (concat (if (not (zerop (length height)))
1120 (format "[%s]" height))
1121 (format "{%s}"
1122 loop-description)))
1123 (end-of-line)
1124 (newline-and-indent)
1125 (newline-and-indent)
1126 (insert "\\forallinend")))
1127
1128 (defun TeX-mac-forever (macro)
1129 "Insert \\forever-block with all arguments.
1130 This is only the optional height"
1131 (let ((height (read-string "Height (optional): ")))
1132 (insert (if (not (zerop (length height)))
1133 (format "[%s]" height)))
1134 (end-of-line)
1135 (newline-and-indent)
1136 (newline-and-indent)
1137 (insert "\\foreverend")))
1138
1139 (defun TeX-mac-ifthenelse (macro)
1140 "Insert \\ifthenelse with all arguments.
1141 These are optional height and the required arguments
1142 left slope, right slope, condition, and the possible
1143 values of the condition"
1144 (let ((height (read-string "Height (optional): "))
1145 (lslope (read-string "Left slope: "))
1146 (rslope (read-string "Right slope: "))
1147 (condition (read-string "Condition: "))
1148 (conditionvl (read-string "Condition value left: "))
1149 (conditionvr (read-string "Condition value right: ")))
1150 (insert (concat (if (not (zerop (length height)))
1151 (format "[%s]" height))

```

```

1152             (format "{%s}{%s}{%s}{%s}{%s}"
1153                   lslope rslope condition conditionvl
1154                   conditionvr)))
1155 (end-of-line)
1156 (newline-and-indent)
1157 (newline-and-indent)
1158 (insert "\\change")
1159 (end-of-line)
1160 (newline-and-indent)
1161 (newline-and-indent)
1162 (insert "\\ifend"))
1163
1164 (defun TeX-mac-inparallel (macro)
1165 "Insert \\inparallel with all arguments, the needed \\task(s)
1166 and the final \\inparallelend. These are optional height and the
1167 required arguments number of tasks and the descriptions for the
1168 parallel tasks"
1169 (let ((height (read-string "Height (optional): "))
1170       (number (read-string "Number of parallel tasks: "))
1171       (text (read-string "Task no. 1: "))
1172       (count 1)
1173       )
1174 (setq number-int (string-to-number number))
1175 (insert (concat (if (not (zerop (length height)))
1176                 (format "[%s]" height)
1177                 (format "{%s}{%s}" number text))))
1178 (while (< count number-int)
1179 (end-of-line)
1180 (newline-and-indent)
1181 (newline-and-indent)
1182 (setq prompt (format "Task no. %d: " (+ 1 count)))
1183 (insert (format "\\task{%s}" (read-string prompt)))
1184 (setq count (1+ count)))
1185 (end-of-line)
1186 (newline-and-indent)
1187 (newline-and-indent)
1188 (insert "\\inparallelend")))
1189
1190 (defun TeX-mac-until (macro)
1191 "Insert \\until with all arguments.
1192 These are the optional height and the required argument condition"
1193 (let ((height (read-string "Height (optional): "))
1194       (condition (read-string "Condition: "))
1195       )
1196 (insert (concat (if (not (zerop (length height)))
1197                 (format "[%s]" height)
1198                 (format "{%s}" condition))))
1199 (end-of-line)
1200 (newline-and-indent)
1201 (newline-and-indent)
1202 (insert "\\untilend")))
1203 (defun TeX-mac-while (macro)
1204 "Insert \\while with all arguments.
1205 These are the optional height and the required argument condition"

```

```

1206 (let ((height (read-string "Height (optional): ")
1207         (condition (read-string "Condition: ")))
1208     (insert (concat (if (not (zerop (length height)))
1209                     (format "[%s]" height))
1210             (format "{-%s-}" condition)))
1211     (end-of-line)
1212     (newline-and-indent)
1213     (newline-and-indent)
1214     (insert "\\whileend")))
1215
1216 (defvar LaTeX-struktex-package-options '("anygradient" "curves" "debug"
1217     "draft" "emlines" "final"
1218     "fixedindent" "nofiller"
1219     "outer" "pict2e" "verification")
1220 "Package options for the struktex package.")
1221
1222 ;;; struktex.el ends here.
1223 </auctex>

```

## Literatur

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