

struktex.sty*

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

gedruckt am 9. Juli 2017

Zusammenfassung

Dieser Artikel beschreibt den Einsatz und die Implementation der *L^AT_EX*-package `struktex.sty` zum Setzen von Struktogrammen nach Nassi-Shneiderman.

Inhaltsverzeichnis

1	Lizenzvereinbarung	1	5	Beispieldatei zum Einbinden in die Dokumentation	24
2	Vorwort	2			
3	Hinweise zur Pflege und Installation sowie die Treiberdatei zur Erzeugung dieser Dokumentation	3	6	Verschiedene Beispieldateien	24
4	Die Benutzungsschnittstelle	6			
4.1	Spezielle Zeichen und Textdarstellung	7	6.1	Beispieldatei Nr. 1	24
4.2	Makros zur Darstellung von Variablen, Schlüsselwörtern und anderen programmierspezifischen Details	7	6.2	Beispieldatei Nr. 2	25
4.3	Die Makros zur Erzeugung von Struktogrammen	9	6.3	Beispieldatei Nr. 3	26
			6.4	Beispieldatei Nr. 4	30
			7	Makros zur Erstellung der Dokumentation des <code>struktex.sty</code>	32
			8	Makefile	37
			9	Stil Datei zur einfachen Eingabe von Struktogrammen beim Arbeiten mit dem (X)emacs und AUCT_EX	42

*Diese Datei hat die Versionsnummer v2.2b-6-g08f43fa, wurde zuletzt bearbeitet am 2017/07/09, und die Dokumentation datiert vom 2017/07/01.

1 Lizenzvereinbarung

This package is copyright © 1995 – 2017 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_ukT_EX 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

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

Mit dem hier beschriebenen Makropaket ist es möglich, Struktogramme mit L^AT_EX zu zeichnen. Das Makropaket wird im folgenden immer St_ukT_EX genannt. Es ist in der Lage, die wichtigsten Elemente eines Struktogrammes wie z. B. Verarbeitungsböcke, Schleifenkonstrukte, Sinnbilder für Alternativen usw. zu generieren. Die Struktogramme werden mit Hilfe der Picture-Umgebung von L^AT_EX erzeugt.¹

Ab Version v-4.1a werden die mathematischen Symbole von $\mathcal{A}\mathcal{M}\mathcal{S}$ -T_EX 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^AT_EX 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

¹ Wer es scheut, Struktogramme mittels L^AT_EX direkt zu schreiben, kann beispielsweise unter <http://strukturizer.fisch.lu/> ein Programm (Strukturizer) finden, mit dem man seine Struktogramme mittels Maus entwickeln und abschließend als L^AT_EX-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.² 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_EX 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.³

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*).

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

³Wenn 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 ]
29 {struktex}
30 \GetFileInfo{struktex.sty}
31
32 \EnableCrossrefs
33 %\DisableCrossrefs    % say \DisableCrossrefs if index is ready
34
35 %\RecordChanges       % say \RecordChanges to gather update information
36
37 %\CodelineIndex       % say \CodelineIndex to index entry code by line number
38
39 \OnlyDescription     % say \OnlyDescription to omit the implementation details
40
41 \MakeShortVerb{\\}   % |\foo| acts like \verb+\foo+
42
43 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
44 % to avoid underfull ... messages while formatting two/three columns
45 \hbadness=10000 \vbadness=10000
46
47 \typeout{\string\primarylanguage: \primarylanguage, \string\language: \the\language}
48
49 \def\languageNGerman{10}          % depends on language.dat, put
50                                     % \the\language here
51
52 \begin{document}

```

```

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

```

4 Die Benutzungsschnittstelle

Der `struktex.sty` wird wie jede andere `.sty`-Datei als *package* in ein \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 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 auen 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

<code>\nat</code>	Wegen ihres häufigen Auftretens sind die Mengen der natürlichen, ganzen, reellen
<code>\integer</code>	und komplexen Zahlen (\mathbb{N} , \mathbb{Z} , \mathbb{R} und \mathbb{C}) im Mathematik-Modus über die folgenden
<code>\real</code>	Makros erreichbar: <code>\nat</code> , <code>\integer</code> , <code>\real</code> und <code>\complex</code> . Ebenso ist das mit
<code>\complex</code>	<code>\emptyset</code> erzeugte „ \emptyset “ als Zeichen für die leere Anweisung auffälliger als das
<code>\emptyset</code>	standardmäßige Zeichen „ \emptyset “. Andere Mengensymbole wie \mathbb{L} (für Lösungsmenge)
	sind über <code>\mathbb{L}</code> zu erzeugen.
<code>\MathItalics</code>	Mit diesen beiden Makros kann die Darstellung von Variablennamen beeinflusst
<code>\MathNormal</code>	werden:

$\textit{NeuerWert} = \textit{AlterWert} + \textit{Korrektur}$	<code>\MathNormal</code>
	<code>\[</code>
	<code>NeuerWert = AlterWert + Korrektur</code>
	<code>\]</code>

und

$\textit{NeuerWert} = \textit{AlterWert} + \textit{Korrektur}$	<code>\MathItalics</code>
	<code>\[</code>
	<code>NeuerWert = AlterWert + Korrektur</code>
	<code>\]</code>

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

<code>\pVariable</code>	Struktogramme enthalten manchmal direkt zu programmierenden Code. Um hier
<code>\pVar</code>	ein einheitliches Aussehen zu erreichen, sind die hier aufgeführten Makros definiert
<code>\pKeyword</code>	worden. Um diese Makros auch in anderem Zusammenhang nutzen zu können,
<code>\pKey</code>	sind sie zu einem eigenen <i>package</i> <code>strukt.p.sty</code> zusammengefasst worden. Dabei
<code>\pComment</code>	wird ab Version 122 zur Darstellung von Code auf das Paket „ <code>url.sty</code> “ 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>&iAdresseEinerVariablen</code>	<code>\pVariable{cEineNormaleVariable}</code>
<code>zZeigerAufEineVariable^.sInhalt</code>	<code>\pVariable{c_eine_normale_Variable}</code>
	<code>\pVariable{&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 *T_EX*-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 \LaTeX . Die in der `picture`-Umgebung gebräuchliche Längeneinheit `\unitlength` wird bei den Struktogrammen nicht benutzt; die Längeneinheit ist aus technischen Gründen auf 1 mm festgelegt. Weiterhin müssen alle Längenangaben ganzzahlige Werte sein. `\unitlength` hat zwar nach dem Zeichnen eines Struktogramms mit \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

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

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 70mm vorgegeben, die Höhe mit 12mm. Eine Alternative ist durch die `centernss`-Umgebung gegeben, die auf Seite 22 beschrieben wird.

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

1. Versuch

Quadratwurzel von π berechnen und ausgeben

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.

Quadratwurzel von π berechnen und ausgeben

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

```
\begin{declaration}[\langle \textit{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 \textit{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
\descriptionindent
\descriptionwidth
\descriptionsep

```
\description{\langle \textit{Variablenname} \rangle}{\langle \textit{Variablenbeschreibung} \rangle}
```

erzeugt. Dabei ist zu beachten, dass `\langle \textit{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:

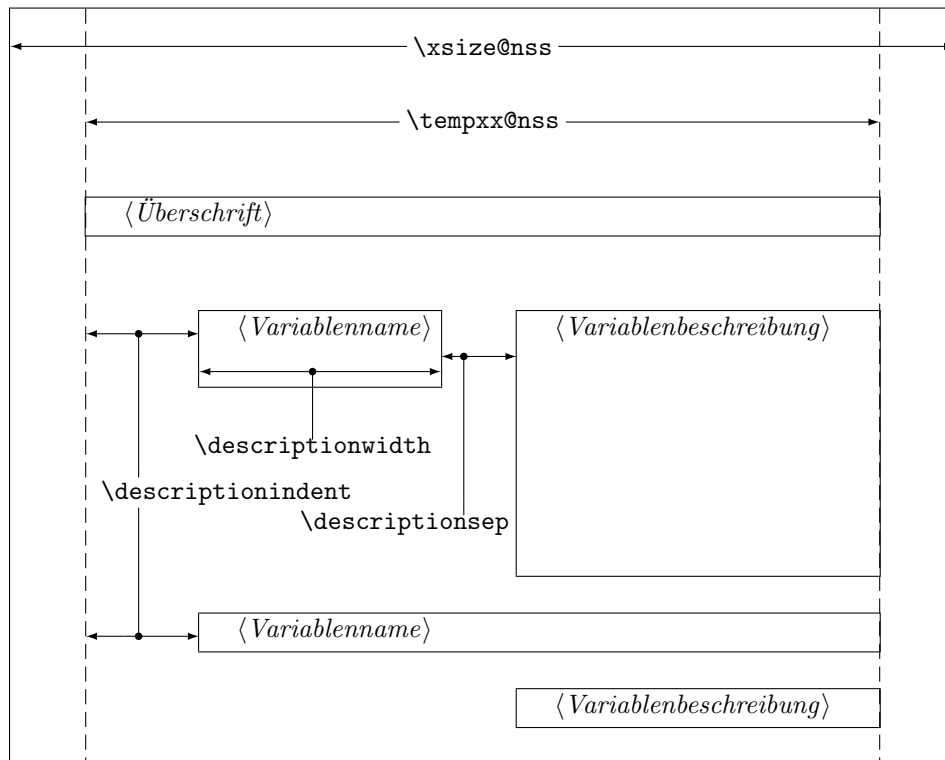


Abbildung 1: Aufbau einer Variablenbeschreibung

```
\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:
`iPar` {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:

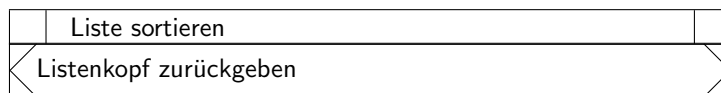
```
\sub[⟨Höhe⟩]{⟨Text⟩}
\return[⟨Höhe⟩]{⟨Text⟩}
```

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⟩\forallin
```

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

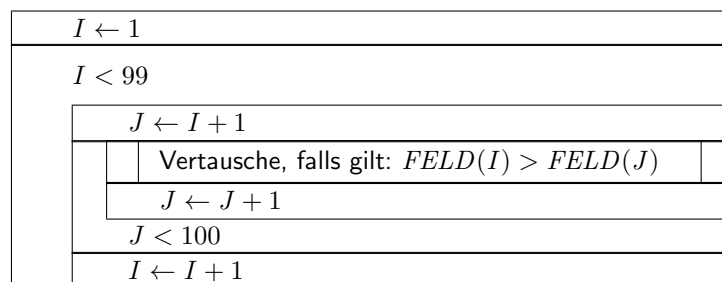
Um Kompatibilität mit der Weiterentwicklung des `struktex.sty` von J. Dietel zu erreichen, gibt es die Makros `\dfr` und `\dfrend` mit derselben Bedeutung wie `\forever` und `\foreverend`.

Die beiden folgenden Beispiele zeigen den Einsatz der `\while`- und `\until`-Makros sowie der `\forallin`-Makros, `\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:

berechne Liste L der ersten 100 Primzahlen
$\forall l \in L$
gib l aus

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 `StylTeX` 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.

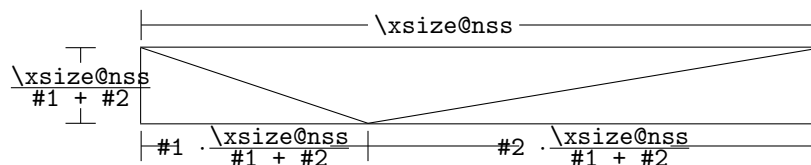
Der If-Then-Else-Befehl sieht so aus:

```

\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.⁴ Die beiden anderen Texte sollten kurz sein (z. B. ja/nein oder true/false), da sie nicht umgebrochen werden können und sonst über ihr

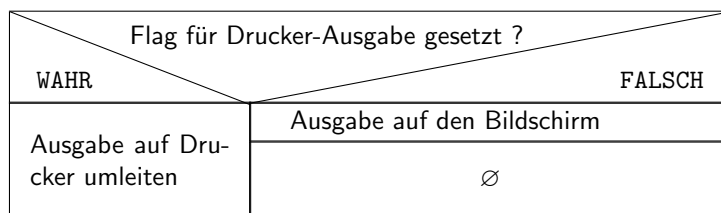
⁴Diese Erweiterung stammt von Daniel Hagedorn, dem ich hiermit herzlich danken möchte

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.⁵ 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}
```

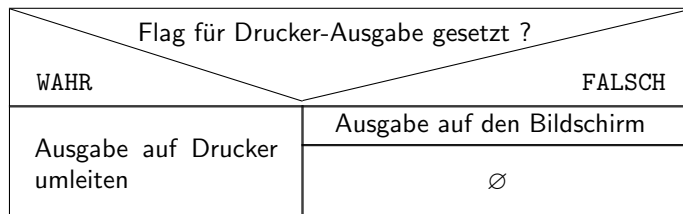
Diese Anweisungen führen zu folgendem Struktogramm:



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:

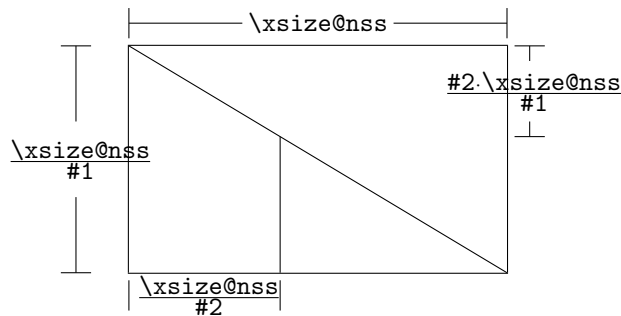
⁵Eventuell ist ein `\strut` hilfreich, um unterschiedliche Höhen von Texten auszugleichen.

```

\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.

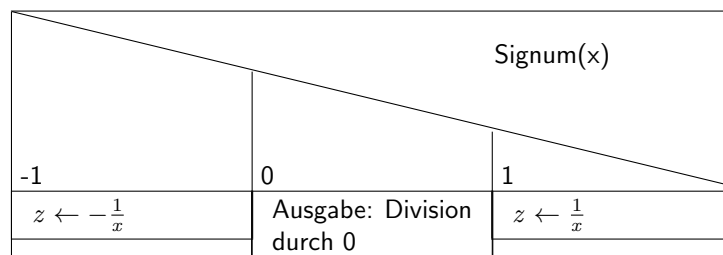
Beispiel 9

```

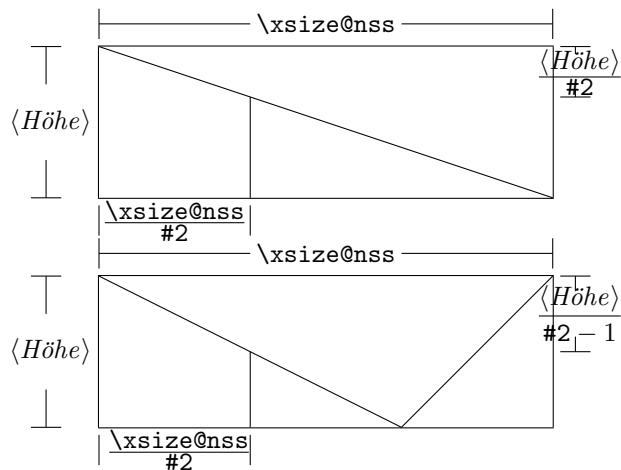
\begin{struktogramm}(95,30)
  \case{4}{3}{Signum(x)}{-1}
    \assign{$z \gets - \frac{1}{x}$}
  \switch{0}
    \assign{Ausgabe: Division durch 0}
  \switch{1}
    \assign{$z \gets \frac{1}{x}$}
  \caseend
\end{struktogramm}

```

Diese Anweisungen führen zu folgendem Struktogramm:



Der optionale Parameter $[\langle Höhe \rangle]$ 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 $[\langle Höhe \rangle]$ 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:

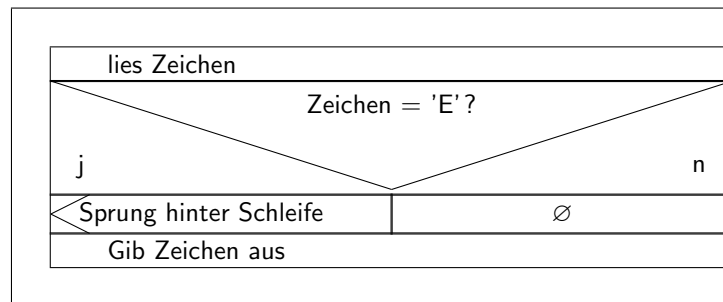
Signum(x)		
-1	1	0
$z \leftarrow -\frac{1}{x}$	$z \leftarrow \frac{1}{x}$	Ausgabe: Division durch 0

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}
```

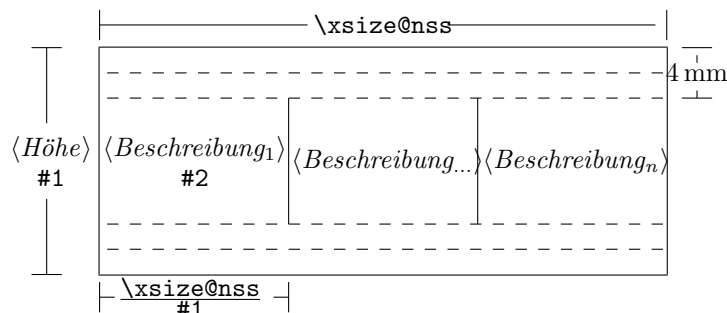
Diese Anweisungen führen zu folgendem Struktogramm:



`\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 `\inparallelend` 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⟩}
\inparallelend
```

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`):

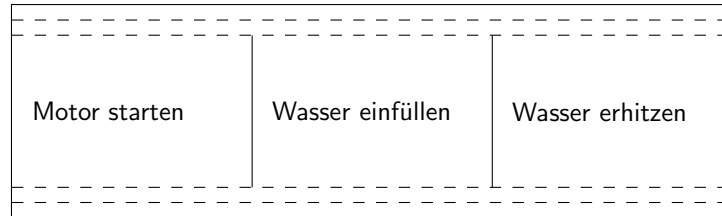


Zu beachten ist, dass die verschiedenen Tasks nicht weiter gegliedert werden dürfen.

Beispiel 13 (Anwendung von `\inparallel`)

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

Diese Anweisungen ergeben das folgende Struktogramm:



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

```

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

Flag für Drucker-Ausgabe gesetzt?	
WAHR	FALSCH
Ausgabe auf Drucker umleiten	Ausgabe auf den Bildschirm
	∅

`\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: Division durch 0	$z \leftarrow \frac{1}{x}$	

`\openstrukt` Diese beiden Makros sind nur der Kompatibilität zu vorherigen Versionen von `StylTeX` willen noch erhalten. Von der Bedeutung her entsprechen sie `\closestrukt` `\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 Beipieldatei zum Einbinden in die Dokumentation

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

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

6 Verschiedene Beipieldateien

6.1 Beipieldatei 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.

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



```

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

```

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

```

```

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

```

6.3 Beispieldatei zum Austesten der Makros des `strukt xp.sty`

Die folgenden Zeilen bilden eine Musterdatei, die zum Austesten der Makros des `strukt xp.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.

```

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

```

```

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

```

```

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

```

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

```



```

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

```

```

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

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

```

706     {\typeout {No file #2.}\endgroup}}
707 </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.

```

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

```

```

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

```

```

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

```

```

855     >> $(PACKAGE).sed; \
856 git describe --long | \
857     xargs git --no-pager show -s --format=format:"      %%% \\\$Author: %an \$\\%n" \
858     >> $(PACKAGE).sed; \
859 printf "%b\\n" "/"' \\\$1" \
860     >> $(PACKAGE).sed; \
861 printf "%b\\n" ";" \
862     >> $(PACKAGE).sed; \
863 printf "%b\\n" "for f in \\\\" \
864     >> $(PACKAGE).sed; \
865 printf "%b\\n" "$(PACKAGE).sty \\\\" \
866     >> $(PACKAGE).sed; \
867 printf "%b\\n" "struktxf.sty \\\\" \
868     >> $(PACKAGE).sed; \
869 printf "%b\\n" "struktxp.sty \\\\" \
870     >> $(PACKAGE).sed; \
871 printf "%b\\n" "strukdoc.sty \\\\" \
872     >> $(PACKAGE).sed; \
873 printf "%b\\n" "; do \\\\" \
874     >> $(PACKAGE).sed; \
875 printf "%b\\n" " set_git_info \\\$f; done" \
876     >> $(PACKAGE).sed; \
877
878 # generate the documentation
879 $(PACKAGE).de.pdf: $(PACKAGE).dtx $(PACKAGE).sty struktex.sed
880 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{<}"
881 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{<}"
882 +mv $(<:.dtx=.pdf) $(<:.dtx=.de.pdf)
883
884 $(PACKAGE).en.pdf: $(PACKAGE).dtx $(PACKAGE).sty
885 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\primarylanguage{english}\input{<}"
886 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\primarylanguage{english}\input{<}"
887 +mv $(<:.dtx=.pdf) $(<:.dtx=.en.pdf)
888
889 # generate the documentation with revision history (only german)
890 history: $(PACKAGE).dtx $(PACKAGE).sty
891 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(HISTORY_OPTIONS)}\input{<}"
892 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(HISTORY_OPTIONS)}\input{<}"
893 +makeindex -s gind.ist $(PACKAGE).idx
894 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
895 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(HISTORY_OPTIONS)}\input{<}"
896
897 # generate the documentation for the developer (revision history always
898 # in german)
899 develop-de: $(PACKAGE).dtx $(PACKAGE).sty
900 +$(PDFLATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)}$(DEVELOPER_OPTIONS)}\input{<}"
901 +$(PDFLATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)}$(DEVELOPER_OPTIONS)}\input{<}"
902 +makeindex -s gind.ist $(PACKAGE).idx
903 +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
904 +$(PDFLATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)}$(DEVELOPER_OPTIONS)}\input{<}"
905 +mv $(<:.dtx=.pdf) $(<:.dtx=.de.pdf)
906
907 develop-en: $(PACKAGE).dtx $(PACKAGE).sty
908 +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}$(DEVELOPER_OPTIONS)}\def\primarylanguage{eng

```



```

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

```

```

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

```

993 <setup>
994 sed -e "s/^ /@/g" | tr '@' '\011' struktex.mk > Makefile
995 </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^ATEX-Dateien. Wenn es eine passende Stildatei für ein L^ATEX-Paket wie das hier vorliegende St^{ru}kTEX 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.

```

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

```

```

1043 "inparallelend"
1044 "sProofOn"
1045 "sProofOff"
1046 '("until" TeX-mac-until)
1047 "untilend"
1048 '("while" TeX-mac-while)
1049 "whileend"
1050 '("return" [ "Height" ] "Return value")
1051 '("sub" [ "Height" ] "Task")
1052 '("CenterNssFile" TeX-arg-file)
1053 '("centernssfile" TeX-arg-file))
1054 (TeX-run-style-hooks
1055 "pict2e"
1056 "emlines2"
1057 "curves"
1058 "struktxp"
1059 "struktxf"
1060 "ifthen")
1061 ;; Filling
1062 ;; Fontification
1063 ))
1064
1065 (defun LaTeX-env-struktogramm (environment)
1066 "Insert ENVIRONMENT with width, height specifications and
1067 optional title."
1068 (let ((width (read-string "Width: "))
1069       (height (read-string "Height: "))
1070       (title (read-string "Title (optional): ")))
1071   (LaTeX-insert-environment environment
1072                             (concat
1073                               (format "(%s,%s)" width height)
1074                               (if (not (zerop (length title)))
1075                                   (format "[%s]" title))))))
1076
1077 (defun LaTeX-env-declaration (environment)
1078 "Insert ENVIRONMENT with an optional title."
1079 (let ((title (read-string "Title (optional): ")))
1080   (LaTeX-insert-environment environment
1081                             (if (not (zerop (length title)))
1082                                 (format "[%s]" title))))))
1083
1084 (defun TeX-mac-case (macro)
1085 "Insert \\case with all arguments, the needed \\switch(es) and
1086 the final \\caseend. These are optional height and the required
1087 arguments slope, number of cases, condition, and the texts for
1088 the different cases"
1089 (let ((height (read-string "Height (optional): "))
1090       (slope (read-string "Slope: "))
1091       (number (read-string "Number of cases: "))
1092       (condition (read-string "Condition: "))
1093       (text (read-string "Case no. 1: "))
1094       (count 1)
1095       )
1096   (setq number-int (string-to-number number))

```

```

1097 (insert (concat (if (not (zerop (length height)))
1098 (format "[%s]" height))
1099 (format "{%s}{%s}{%s}{%s}"
1100 slope number condition text)))
1101 (while (< count number-int)
1102 (end-of-line)
1103 (newline-and-indent)
1104 (newline-and-indent)
1105 (setq prompt (format "Case no. %d: " (+ 1 count)))
1106 (insert (format "\\switch{%s}" (read-string prompt)))
1107 (setq count (1+ count)))
1108 (end-of-line)
1109 (newline-and-indent)
1110 (newline-and-indent)
1111 (insert "\\caseend")))
1112
1113 (defun TeX-mac-forallin (macro)
1114 "Insert \\forallin-block with all arguments.
1115 This is the optional height and the description of the loop"
1116 (let ((height (read-string "Height (optional): "))
1117 (loop-description (read-string "Description of loop: ")))
1118 (insert (concat (if (not (zerop (length height)))
1119 (format "[%s]" height))
1120 (format "{%s}"
1121 loop-description))))
1122 (end-of-line)
1123 (newline-and-indent)
1124 (newline-and-indent)
1125 (insert "\\forallinend")))
1126
1127 (defun TeX-mac-forever (macro)
1128 "Insert \\forever-block with all arguments.
1129 This is only the optional height"
1130 (let ((height (read-string "Height (optional): ")))
1131 (insert (if (not (zerop (length height)))
1132 (format "[%s]" height))
1133 (end-of-line)
1134 (newline-and-indent)
1135 (newline-and-indent)
1136 (insert "\\foreverend"))))
1137
1138 (defun TeX-mac-iffthenelse (macro)
1139 "Insert \\iffthenelse with all arguments.
1140 These are optional height and the required arguments
1141 left slope, right slope, condition, and the possible
1142 values of the condition"
1143 (let ((height (read-string "Height (optional): "))
1144 (lslope (read-string "Left slope: "))
1145 (rslope (read-string "Right slope: "))
1146 (condition (read-string "Condition: "))
1147 (conditionvl (read-string "Condition value left: "))
1148 (conditionvr (read-string "Condition value right: ")))
1149 (insert (concat (if (not (zerop (length height)))
1150 (format "[%s]" height))

```

```

1151             (format "%s{%s}{%s}{%s}{%s}"
1152                     lslope rslope condition conditionvl
1153                     conditionvr)))
1154 (end-of-line)
1155 (newline-and-indent)
1156 (newline-and-indent)
1157 (insert "\\change")
1158 (end-of-line)
1159 (newline-and-indent)
1160 (newline-and-indent)
1161 (insert "\\ifend"))))
1162
1163 (defun TeX-mac-inparallel (macro)
1164   "Insert \\inparallel with all arguments, the needed \\task(s)
1165   and the final \\inparallelend. These are optional height and the
1166   required arguments number of tasks and the descriptions for the
1167   parallel tasks"
1168   (let ((height (read-string "Height (optional): "))
1169         (number (read-string "Number of parallel tasks: "))
1170         (text (read-string "Task no. 1: "))
1171         (count 1)
1172         )
1173     (setq number-int (string-to-number number))
1174     (insert (concat (if (not (zerop (length height)))
1175                       (format "[%s]" height)
1176                       (format "%s{%s}" number text))
1177               (while (< count number-int)
1178                 (end-of-line)
1179                 (newline-and-indent)
1180                 (newline-and-indent)
1181                 (setq prompt (format "Task no. %d: " (+ 1 count)))
1182                 (insert (format "\\task%s" (read-string prompt)))
1183                 (setq count (1+ count)))
1184               (end-of-line)
1185               (newline-and-indent)
1186               (newline-and-indent)
1187               (insert "\\inparallelend"))))
1188
1189 (defun TeX-mac-until (macro)
1190   "Insert \\until with all arguments.
1191   These are the optional height and the required argument condition"
1192   (let ((height (read-string "Height (optional): "))
1193         (condition (read-string "Condition: "))
1194         )
1195     (insert (concat (if (not (zerop (length height)))
1196                       (format "[%s]" height)
1197                       (format "%s" condition))
1198               (end-of-line)
1199               (newline-and-indent)
1200               (newline-and-indent)
1201               (insert "\\untilend"))))
1202
1203 (defun TeX-mac-while (macro)
1204   "Insert \\while with all arguments.
1205   These are the optional height and the required argument condition"

```

```

1205 (let ((height (read-string "Height (optional): "))
1206         (condition (read-string "Condition: ")))
1207   (insert (concat (if (not (zerop (length height)))
1208                       (format "[%s]" height))
1209                 (format "{-%s-}" condition)))
1210   (end-of-line)
1211   (newline-and-indent)
1212   (newline-and-indent)
1213   (insert "\\whileend"))
1214
1215 (defvar LaTeX-struktex-package-options '("curves" "draft" "emlines" "final"
1216                                           "pict2e" "anygradient" "verification"
1217                                           "nofiller" "debug" "outer")
1218   "Package options for the struktex package.")
1219
1220 ;;; struktex.el ends here.
1221 </auctex>

```

Literatur

- [Fut89] Gerald Futschek. *Programmentwicklung und Verifikation*. Springer Verlag, Wien – New York, 1989.
- [GMS94] Michel Goossens, Frank Mittelbach and Alexander Samarin. *The L^AT_EX Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.
- [GMS04] Frank Mittelbach and Michel Goossens. *The L^AT_EX Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, second edition, 2004.
- [Knu86] D. E. Knuth. *The T_EX-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_EXnische Komödie*, 8(4):23–40, Februar 1996.
- [Rah92] Sebastian Rahtz. *The oz package*, 1992.