struktex.sty*

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

gedruckt am 15. Juni 2005

Zusammenfassung

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

Inhaltsverzeichnis

1 Vorwort

Mit dem hier beschriebenen Makropaket ist es möglich, Struktogramme mit LaTeX zu zeichnen. Das Makropaket wird im folgenden immer StruktTeX genannt. Es ist

*Diese Datei hat die Versionsnummer v8.0f, wurde zuletzt bearbeitet am 2005/05/17, und die Dokumentation datiert vom 2005/05/17.

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 LaTeX erzeugt.

Ab Version 4.1a werden die mathematischen Symbole von $\mathcal{AMS}$-TeX 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 ($\varnothing$) 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, Variablennamen 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 LaTeX gegebenen Einschränkungen – es gibt nur vordefinierte Steigungen – aufzuheben. Dies ist – für das `\ifthenelse` mit den Versionen 4.1a und 4.1b, für das `\switch` mit der Version 4.2a – erledigt, nicht jedoch 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 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 8.0a realisiert worden.

Weitere Zukunftspläne sind:

1. Ein `\otherwise`-Zweig beim `\switch` (abgeschlossen durch die Version 4.2a).
2. Die Neuimplementation der `declaration`-Umgebung mittels der `list`-Umgebung gemäß [**?**, Abs. 3.3.4] (abgeschlossen mit der Version 4.5a).
3. Die Anpassung an LaTeX 2_ε im Sinne eines Packages (abgeschlossen durch die Version 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 4.5a).
6. die vollständige Implementation der Makros `\pVar`, `\pKey`, `\pFonts`, `\pTrue`, `\pFalse` und `\pBoolValue` (erledigt vor Version 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 4.4a.
8. Die Unabhängigkeit der Dokumentation von anderen .sty-Dateien wie etwa dem JHfMakro.sty (abgeschlossen mit der Version 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.

2 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.

```
latex 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 7).

Die Dokumentation wird wie üblich durch

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

erzeugt.[1] Das Ergebnis dieses Formatierlaufes ist die Dokumentation in Form einer .dvi-Datei, die in gewohnter Weise weiterbearbeitet werden kann. Weitere Informationen zum Arbeiten mit der integrierten Dokumentation findet man in [Mit01] und [MDB01]. Die Dateien tst_strf.tex, tst_strp.tex schließlich sind Dateien zum Austesten der hier beschriebenen Makros.

Die Installation wird abgeschlossen, indem die Datei `struktex.sty` in ein Verzeichnis verschoben wird, das von TEX gefunden werden kann, das ist in einer

[1] Die Erzeugung der Dokumentation kann durch den Einsatz einer `make`-Datei vereinfacht werden, vgl. Abschnitt 7

TDS-konformen Installation typischerweise `.../tex/latex/struktex/`, die Dokumentation wird analog in das Verzeichnis `.../doc/latex/struktex/` verschoben.[2]

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*).

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

```
\documentclass[a4paper, english, ngerman]{ltxdoc}

\usepackage{babel}                  % for switching the documentation language
\usepackage{strukdoc}               % the style-file for formatting this
                                    % documentation
\usepackage[pict2e]% <-------------- to produce finer results
        {struktex}                  % visible under xdvi, alternative
                                    % curves or emlines2 (visible only under
                                    % ghostscript), leave out if not
                                    % available

\GetFileInfo{struktex.sty}

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

%\RecordChanges      % say \RecordChanges to gather update information

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

\OnlyDescription    % say \OnlyDescription to omit the implementation details

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

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

\def\languageNGerman{3}

\begin{document}
\makeatletter
\@ifundefined{selectlanguageEnglish}{}{\selectlanguage{english}}
```

[2] Wenn die automatische Installation (vgl. Abschnitt 7) vorgenommen wird, erfolgt diese in die Verzeichnisse `.../doc/latex/jhf/struktex/`, `.../tex/latex/jhf/struktex/` und `.../source/latex/jhf/struktex/`.

```
\makeatother
\DocInput{struktex.dtx}
\end{document}
```

3 Die Benutzungsschnittstelle

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

```
\usepackage{struktex}
```

Das Laden des Paketes kann unter Angabe einer zusätzlichen Option geschehen, um beliebige Steigungen in Struktogrammen zeichnen zu können. Diese Option kann `emlines`, `curves`, oder `pict2e` lauten; ersteres 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.

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.

3.1 Spezielle Zeichen und Textdarstellung

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

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

$$NeuerWert = AlterWert + Korrektur$$

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

und

$$NeuerWert = AlterWert + Korrektur$$

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

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

\pVariable
\pVar
\pKeyword
\pKey
\pComment

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:

```
cEineNormaleVariable
c_eine_normale_Variable
&iAdresseEinerVariablen
zZeigerAufEineVariable^.sInhalt
```

```
\obeylines
 \renewcommand{\pLanguage}{C}
 \pVariable{cEineNormaleVariable}
 \pVariable{c_eine_normale_Variable}
 \pVariable{&iAdresseEinerVariablen}
 \renewcommand{\pLanguage}{Pascal}
 \pVariable{zZeigerAufEineVariable^.sInhalt}
```

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:

```
begin
program
#include
```

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

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
\pFalse
\pFonts
\pBoolValue

Logische Werte spielen in der Programmierung eine wesentliche Rolle. Mit `\pTrue` und `\pFalse` sind entsprechende Werte vorgegeben: `WAHR` und `FALSCH`.

Der Makro `\pFonts` dient der Auswahl von Fonts zur Darstellung von Variablen, 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` umdefiniert werden. Somit liefern die Zeilen

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

das folgende Ergebnis:

nein = `not` *ja*

\sVar
\sKey
\sTrue
\sFalse

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

3.3 Die Makros zur Erzeugung von Struktogrammen

struktogramm
\sProofOn
\sProofOff
\PositionNSS

Die Umgebung

```
\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 Struktogrammes mit StruktTeX den gleichen Wert wie vorher, ist aber innerhalb eines Struktogrammes umdefiniert 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

```
\assign[⟨Höhe⟩]{⟨Inhalt⟩},
```

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 zentriert in den Kasten gesetzt; ist er dafür zu lang, so wird ein Paragraph gesetzt.

Beispiel 1

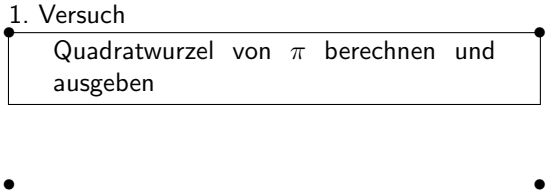
Ein einfaches Struktogramm wird mit den folgenden Anweisungen erzeugt:

```
\sProofOn
\begin{struktogramm}(70,20)[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 18 beschrieben wird.

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



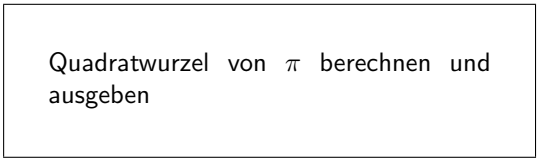
Die Bedeutung des optionalen Argumentes macht das folgende Beispiel deutlich.

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}[⟨*Überschrift*⟩]

...

\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{⟨*Überschrift*⟩} global festgelegt. Will man für ein einzelnes Struktogramm einen speziellen Titel erzeugen, so gibt man diesen in den eckigen Klammern an.

\description
\descriptionindent
\descriptionwidth
\descriptionsep

Innerhalb der declaration-Umgebung werden die Beschreibungen der einzelnen Variablen mit

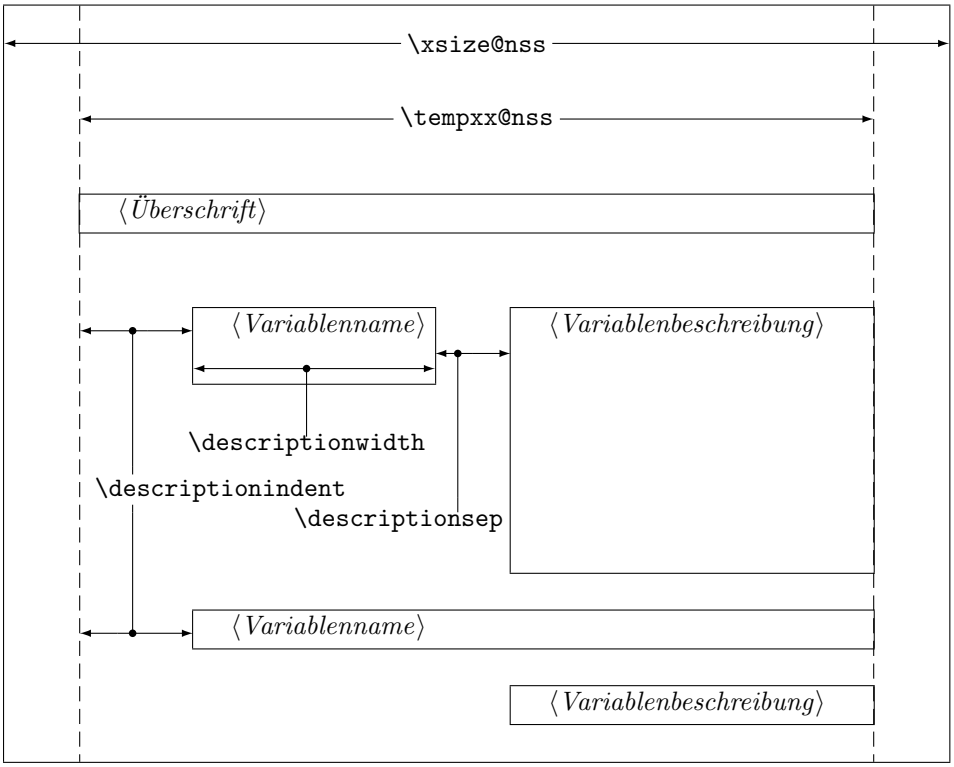


Abbildung 1: Aufbau einer Variablenbeschreibung

`\description{`⟨*Variablenname*⟩`}{`⟨*Variablenbeschreibung*⟩`}`

erzeugt. Dabei ist zu beachten, dass ⟨*Variablenname*⟩ keine schließende eckige Klammer „]“ beinhalten darf, da dieser Makro mittels des `\item`-Makros definiert worden ist. Eckige Klammer 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 StrukTEX vorgegeben werden). Die Vorbesetzung dieser Werte ist folgendermaßen:

```
\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}
```

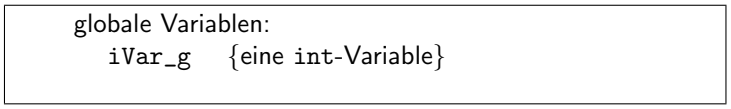
Das ergibt:

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)
    {\catcode`\_=12%
        \assign{%
            \begin{declaration}
                \description{\pVar{iVar_g}}{eine \pKey{int}-Variable}
            \end{declaration}
        }
    }
\end{struktogramm}
```

Dies ergibt das folgende Aussehen:



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 TEX nicht aus.

\sub
\return

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

\sub[⟨*Höhe*⟩]{⟨*Text*⟩}

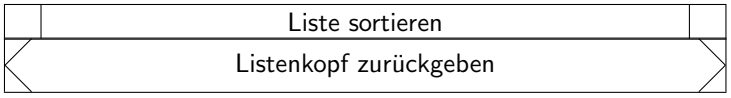
\return[⟨*Höhe*⟩]{⟨*Text*⟩}

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
\whileend
\until
\untilend
\forever
\foreverend

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

`\while[`⟨*Breite*⟩`]{`⟨*Text*⟩`}`⟨*Unterstruktogramm*⟩`\whileend`

`\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

An Stelle von ⟨*Unterstruktogramm*⟩ können beliebige Befehle von StrukTEX stehen (mit Ausnahme von `\openstrukt` und `\closestrukt`), die das Struktogramm innerhalb der `\while`-, der `\until`- oder der `\forever`-Schleife bilden.

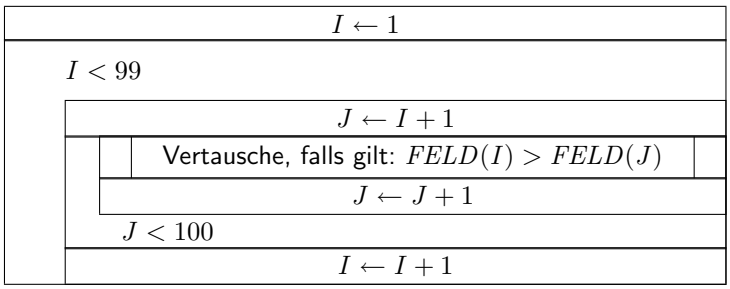
Um Kompatiblität mit der Weiterentwicklung des struktex.sty von J. Dietel zu erreichen, gibt es die Makros `\dfr` und `\dfrend` mit derselben Bedeutung wie `\forever` und `\foreverend`.

Die folgenden Beispiele zeigen den Einsatz der `\while`- und `\until`-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:



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

`\ifthenelse` `\change` `\ifend`

Zur Darstellung von Alternativen stellt StrukTEX die Sinnbilder für einen If-Then-Else-Block und für mehrfache Alternativen eine Case-Konstruktion zur Verfügung. Da in der Picture-Umgebung von LaTeX nur Linien mit bestimmten

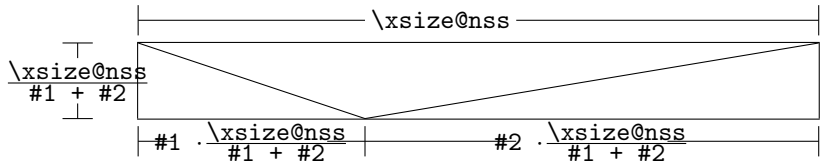
Steigungen gezeichnet werden können, muss der Benutzer bei beiden Befehlen selbst den Winkel bestimmen, mit dem die notwendigen schrägen Linien gezeichnet werden sollen (hier ist also etwas mehr ‚Handarbeit' nötig).

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

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

```
\ifthenelse[⟨Höhe⟩]{⟨Linker Winkel⟩}{⟨Rechter Winkel⟩}
    {⟨Bedingung⟩}{⟨Linker Text⟩}{⟨Rechter Text⟩}
  ⟨Unterstruktogramm⟩
\change
  ⟨Unterstruktogramm⟩
\ifend
```

Für den Fall, dass das optionale Argument ⟨*Höhe*⟩ nicht angegeben ist, sind ⟨*Linker Winkel*⟩ (`#1`) und ⟨*Rechter Winkel*⟩ (`#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 ⟨*Höhe*⟩ vorgegeben, so bestimmt dieser Wert statt des Ausdruckes $\frac{\texttt{\textbackslash xsize@nss}}{\texttt{\#1 + \#2}}$ die Höhe des Bedingungsrechteckes.



⟨*Bedingung*⟩ wird in das so gebildete obere mittlere Dreieck gesetzt; die Parameter ⟨*Linker Text*⟩ und ⟨*Rechter Text*⟩ werden in das linke bzw. rechte untere Dreieck gesetzt. Der Bedingungs-Text kann in seinem Dreiecks-Feld umgebrochen werden. Ab Version 5.3 wird der Bedingungstext durch geeigneten Umbruch beliebigen Steigungen angepasst.[3] Die beiden anderen Texte sollten kurz sein (z. B. ja/nein oder true/false), da sie nicht umgebrochen werden können und sonst über ihr 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 ∅ ergänzt.[4] Mit `\ifend` wird das If-Then-Else-Element beendet. Es folgen zwei Beispiele für die Anwendung.

Beispiel 6

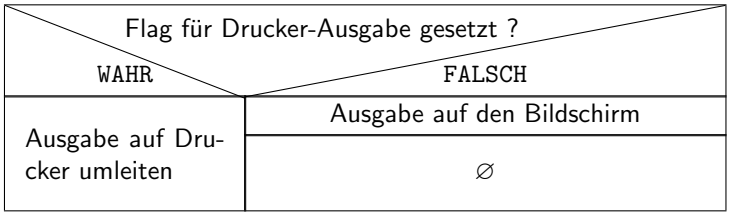
```
\begin{struktogramm}(95,32)
    \ifthenelse[12]{1}{2}
        {Flag f"ur Drucker-Ausgabe gesetzt ?}{\sTrue}{\sFalse}
        \assign[15]{Ausgabe auf Drucker umleiten}
```

[3] Diese Erweiterung stammt von Daniel Hagedorn, dem ich hiermit herzlich danken möchte

[4] Eventuell ist ein `\strut` hilfreich, um unterschiedliche Höhen von Texten auszugleichen.

```
    \change
        \assign{Ausgabe auf den Bildschirm}
    \ifend
\end{struktogramm}
```

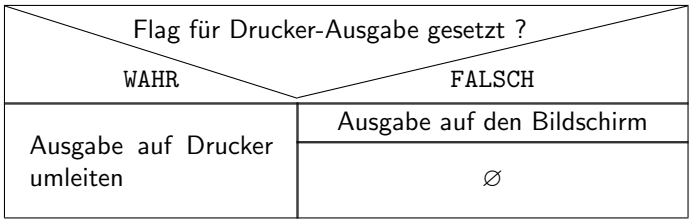
Diese Anweisungen führen zu folgendem Struktogramm:



Beispiel 7

```
\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:

\case[⟨*Höhe*⟩]{⟨*Winkel*⟩}{⟨*Anzahl der Fälle*⟩}{⟨*Bedingung*⟩}{⟨*Text des 1. Falles*⟩}

⟨*Unterstruktogramm*⟩

\switch[⟨*Position*⟩]{⟨*Text des 2. Falles*⟩}

⟨*Unterstruktogramm*⟩

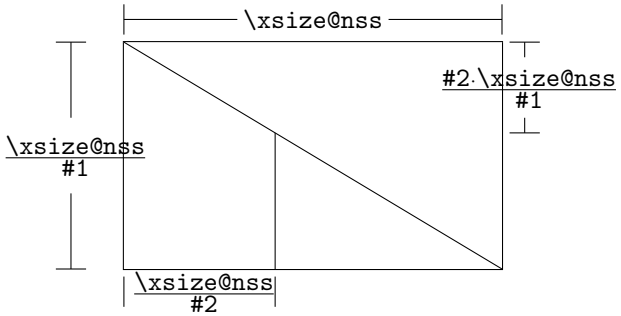
...

\switch[⟨*Position*⟩]{⟨*Text des n. Falles*⟩}

⟨*Unterstruktogramm*⟩

\caseend

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

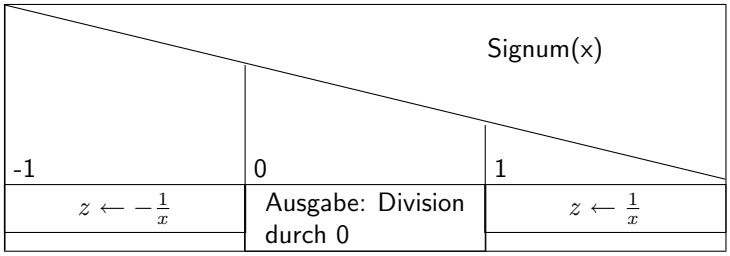


Der zweite Parameter ⟨*Anzahl der Fälle*⟩ gibt die Anzahl der zu zeichnenden Fälle an; alle Unterstruktogramme der einzelnen Fälle erhalten die gleiche Breite. Der ⟨*Text des 1. Falles*⟩ muss als Parameter des `\case`-Befehls angegeben werden, alle weiteren Fälle werden über den `\switch`-Befehl eingeleitet. 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 8

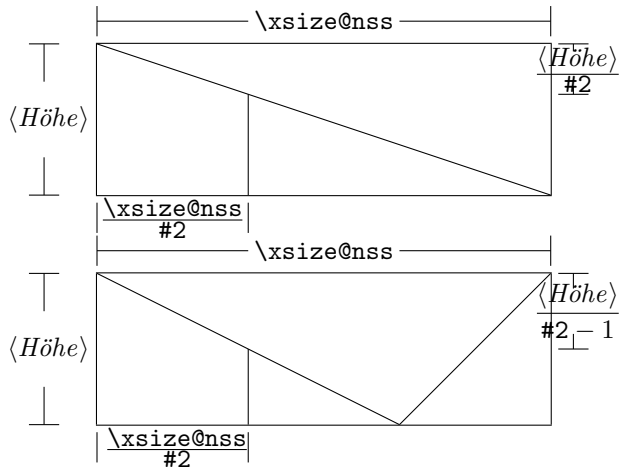
```
\begin{struktogramm}(95,30)
    \case{4}{3}{Signum(x)}{-1}
        \assign{$z \gets - \frac{1}{x}$}
    \switch{0}
        \assign{Ausgabe: Division durch 0}
    \switch{1}
        \assign{$z \gets \frac{1}{x}$}
    \caseend
\end{struktogramm}
```

Diese Anweisungen führen zu folgendem Struktogramm:



Der optionale Parameter `[`⟨*Höhe*⟩`]` 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 `[`⟨*Höhe*⟩`]` führt zu einer anderen Bedeutung von ⟨*Winkel*⟩. Ist

der Wert gerade, wird wie zuvor eine gerade Linie zur Aufteilung des zugrundeliegenden Rechtecks gezeichnet; ist der Wert hingegegen ungerade, wird der letzte Fall wie im folgenden gezeigt als Sonderfall gezeichnet.



Beispiel 9

```
\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:

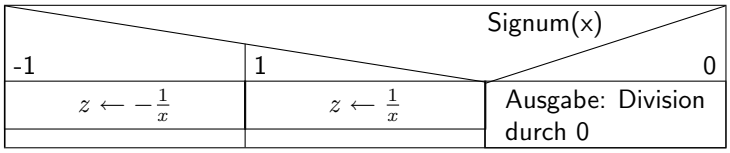
Signum(x)		
-1	0	1
$z \leftarrow -\frac{1}{x}$	Ausgabe: Division durch 0	$z \leftarrow \frac{1}{x}$

Ist der erste Parameter hingegen ungerade, wird ein Standardzweig gezeichnet; der Wert für den Standardfall sollte dann rechtsbündig ausgerichtet werden.
Beispiel 10

```
\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:

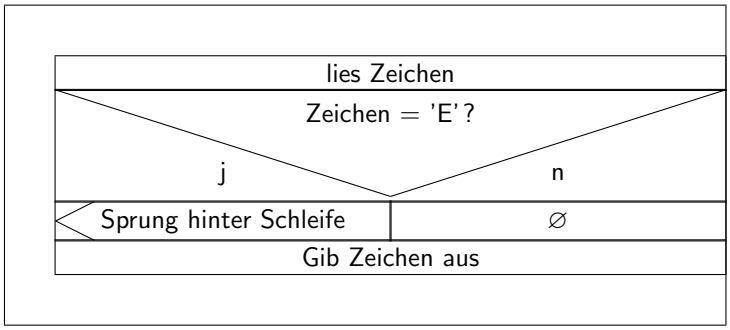


Das folgende Beispiel zeigt, wie mittels einfacher Verzweigung aus einer Endlosschleife gesprungen werden kann. Das Beispiel lässt sich ohne weiteres auf eine mehrfache Verzweigung übertragen.

Beispiel 11

```
\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:



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

```
\begin{centernss}
```

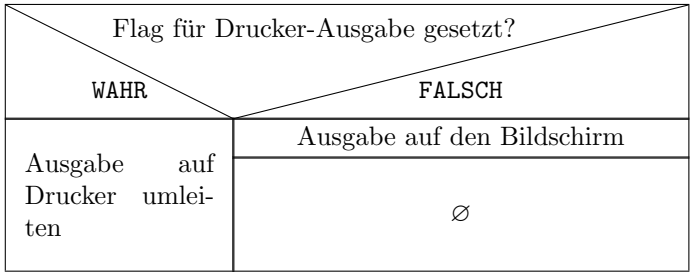
⟨*Struktogramm*⟩

```
\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:



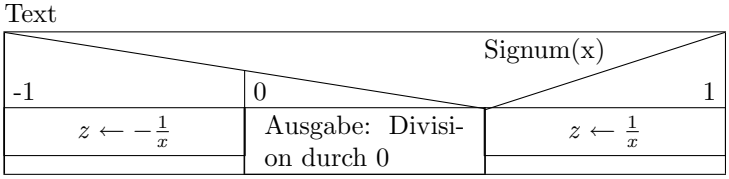
`\CenterNssFile` Häufig gibt es den Fall, dass Struktogramme in eigenen Dateien abgelegt werden, damit sie für sich allein auf Korrektheit getestet werden können oder in anderen Zusammenhängen genutzt werden können. Sollen sie zentriert eingebunden werden, kann nicht mit der folgenden Konstruktion gearbeitet 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` definert 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 4, Zeile 2–10 gezeigte Aussehen hat, so führt die Anweisung

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

zu folgendem Aussehen des formatierten Textes:



`\openstrukt` `\closestrukt`

Diese beiden Makros sind nur der Kompatibilität zu vorherigen Versionen von StruktTEX willen noch erhalten. Von der Bedeutung her entsprechen sie `\struktogramm` und `\endstruktogramm`. Die Syntax ist

`\openstrukt{`⟨*width* ⟩`}{`⟨*height* ⟩`}`

und

`\closestrukt`.

4 Beipieldatei zum Einbinden in die Dokumentation

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

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

5 Verschiedene Beispieldateien

5.1 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.

```
\documentclass{article}
\usepackage{struktex}

\begin{document}

%\sProofOn{}
\begin{struktogramm}(90,137)
    \assign%
```

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

\end{document}
```

5.2 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.

```
\documentclass{article}
\usepackage[pict2e, verification]{struktex}

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

```
        \assign[\StruktBoxHeight]{\(a\gets c\)}
        \assign[\StruktBoxHeight]{\(c\gets tmp\)}
        \assert[\StruktBoxHeight]{\(a<c\)}
    \ifend
    \assert[\StruktBoxHeight]{\(a\le c\)}
    \ifthenelse[\StruktBoxHeight]{2}{1}{\(a\le b\)}{j}{n}
        \assert[\StruktBoxHeight]{\(a\le b \wedge a\le c\)}
        \ifthenelse[\StruktBoxHeight]{1}{1}{\(b\le c\)}{j}{n}
            \assert[\StruktBoxHeight]{\(a\le b \le c\)}
        \change
            \assert[\StruktBoxHeight]{\(a \le  c<b\)}
            \assign[\StruktBoxHeight]{\(tmp\gets c\)}
            \assign[\StruktBoxHeight]{\(c\gets b\)}
            \assign[\StruktBoxHeight]{\(b\gets tmp\)}
            \assert[\StruktBoxHeight]{\(a\le b<c\)}
        \ifend
    \change
        \assert[\StruktBoxHeight]{\(b < a\le c\)}
        \assign[\StruktBoxHeight]{\(tmp\gets a\)}
        \assign[\StruktBoxHeight]{\(a\gets b\)}
        \assign[\StruktBoxHeight]{\(b\gets tmp\)}
        \assert[\StruktBoxHeight]{\(a<b\le c\)}
    \ifend
    \assert[\StruktBoxHeight]{\(a\le b \le c\)}
\end{struktogramm}

\end{document}
```

5.3 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.

```
\documentclass{article}

\usepackage{struktxp,struktxf}

\nofiles

\begin{document}

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

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

\paragraph{After changing the boolean values with}
```

```
\verb-\pBoolValue{yes}{no}-:

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

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

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

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

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

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

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

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

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

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

```
200
201 \paragraph{After changing the boolean values with}
202 \verb-\pBoolValue{\texttt{yes}}{\texttt{no}}-:
203
204 {\obeylines
205 \pBoolValue{\texttt{yes}}{\texttt{no}}
206 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
207 }
208
209 \paragraph{after changing the fonts with}
210 \verb-\pFonts{\itshape}{\sffamily\bfseries}{}-:
211
212 {\obeylines
213 \pFonts{\itshape}{\sffamily\bfseries}{}
214 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
215 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
216 in math mode: \(\pVar{a}+\pVar{iV_g}\)
217 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
218 }
219
220 \paragraph{after changing the fonts with}
221 \verb-\pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}-:
222
223 {\obeylines
224 \pFonts{\ttfamily}{\ttfamily\bfseries}{\ttfamily\slshape}
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 fonts with}
232 \verb-\pFonts{\itshape}{\bfseries\itshape}{}-:
233
234 {\obeylines
235 \pFonts{\itshape}{\bfseries\itshape}{}
236 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
237 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
238 in math mode: \(\pVar{a}+\pVar{iV_g}\)
239 boolean values: \sTrue, \sFalse, \pTrue, \pFalse
240
241 \vspace{15pt}
242 Without \textit{italic correction}:
243     M \pVar{M} M \pKey{M} M. \pVar{M}. M. \pKey{M}. M.
244 }
245
246 \pLanguage{Java}
247 \pBoolValue{\texttt{WAHR}}{\texttt{FALSCH}}
248 \section*{Default values (Java):}
249
250 {\obeylines
251 Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
252 Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
253 in math mode: \(\pVar{a}+\pVar{iV_g}\)
```

```
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

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

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

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

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

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

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

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

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

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

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

{\obeylines
Variables: \pVar{iV_g}, \sVar{zV^}, \pVariable{&i}
```

```
Keywords: \pKeyword{begin}, \pKey{while}, \sKey{__CPP__}
in math mode: \(\pVar{a}+\pVar{iV_g}\)
boolean values: \sTrue, \sFalse, \pTrue, \pFalse
}

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

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

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

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

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

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

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

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

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

\end{document}
%%
%% End of file `struktex-test-2.tex'.
```

6 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.

```
⟨*strukdoc⟩
\RequirePackage{ifpdf}
\ProvidesPackage{strukdoc}
             [\filedate\space\fileversion\space (Jobst Hoffmann)]
\newif\ifcolor \IfFileExists{color.sty}{\colortrue}{}
\ifpdf \RequirePackage[colorlinks]{hyperref}\else
    \def\href#1{\texttt}\fi
\ifcolor \RequirePackage{color}\fi
\RequirePackage{nameref}
\RequirePackage{url}
\renewcommand\ref{\protect\T@ref}
\renewcommand\pageref{\protect\T@pageref}
\@ifundefined{zB}{}{\endinput}
\providecommand\pparg[2]{%
  {\ttfamily(}\meta{#1},\meta{#2}{\ttfamily)}}
\providecommand\envb[1]{%
  {\ttfamily\char`\\begin\char`\{#1\char`\}}}
\providecommand\enve[1]{%
  {\ttfamily\char`\\end\char`\{#1\char`\}}}
\newcommand{\zBspace}{z.\,B.}
\let\zB=\zBspace
\newcommand{\dhspace}{d.\,h.}
\let\dh=\dhspace
\let\foreign=\textit
\newcommand\Abb[1]{Abbildung~\ref{#1}}
\def\newexample#1{%
  \@ifnextchar[{\@oexmpl{#1}}{\@nexmpl{#1}}}
\def\@nexmpl#1#2{%
  \@ifnextchar[{\@xnexmpl{#1}{#2}}{\@ynexmpl{#1}{#2}}}
\def\@xnexmpl#1#2[#3]{%
  \expandafter\@ifdefinable\csname #1\endcsname
    {\@definecounter{#1}\@newctr{#1}[#3]%
     \expandafter\xdef\csname the#1\endcsname{%
       \expandafter\noexpand\csname the#3\endcsname \@exmplcountersep
          \@exmplcounter{#1}}%
     \global\@namedef{#1}{\@exmpl{#1}{#2}}%
     \global\@namedef{end#1}{\@endexample}}}
\def\@ynexmpl#1#2{%
```

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

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

\newwrite\struktex@out
\newenvironment{example}%
 {\begingroup% Lets keep the changes local
  \@bsphack
  \immediate\openout \struktex@out \jobname.tmp
  \let\do\@makeother\dospecials\catcode`\^^M\active
  \def\verbatim@processline{%
    \immediate\write\struktex@out{\the\verbatim@line}}%
  \verbatim@start}%
 {\immediate\closeout\struktex@out\@esphack\endgroup%
%
% And here comes the part of Tobias Oetiker
%
  \par\small\addvspace{3ex plus 1ex}\vskip -\parskip
  \noindent
  \makebox[0.45\linewidth][l]{%
  \begin{minipage}[t]{0.45\linewidth}
    \vspace*{-2ex}
    \setlength{\parindent}{0pt}
    \setlength{\parskip}{1ex plus 0.4ex minus 0.2ex}
    \begin{trivlist}
      \item\input{\jobname.tmp}
    \end{trivlist}
  \end{minipage}}%
```

```
  \hfill%
  \makebox[0.5\linewidth][l]{%
  \begin{minipage}[t]{0.50\linewidth}
    \vspace*{-1ex}
    \verbatiminput{\jobname.tmp}
  \end{minipage}}
  \par\addvspace{3ex plus 1ex}\vskip -\parskip
}

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

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

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

```
    \else
      \def\@tempa##1!end\@nil{##1}%
      \@temptokena{!end}%
      \def\next{\expandafter\verbatim@test\@tempa#1\@nil~}%
    \fi \next}%
\def\verbatim@test#1{%
       \let\next\verbatim@test
       \if\noexpand#1\noexpand~%
         \expandafter\verbatim@addtoline
           \expandafter{\the\@temptokena}%
         \verbatim@processline
         \verbatim@startline
         \let\next\verbatim@
       \else \if\noexpand#1
         \@temptokena\expandafter{\the\@temptokena#1}%
       \else \if\noexpand#1\noexpand[%
         \let\@tempc\@empty
         \let\next\verbatim@testend
       \else
         \expandafter\verbatim@addtoline
           \expandafter{\the\@temptokena}%
         \def\next{\verbatim@#1}%
       \fi\fi\fi
       \next}%
\def\verbatim@testend#1{%
     \if\noexpand#1\noexpand~%
       \expandafter\verbatim@addtoline
         \expandafter{\the\@temptokena[}%
       \expandafter\verbatim@addtoline
         \expandafter{\@tempc}%
       \verbatim@processline
       \verbatim@startline
       \let\next\verbatim@
     \else\if\noexpand#1\noexpand]%
       \let\next\verbatim@@testend
     \else\if\noexpand#1\noexpand!%
       \expandafter\verbatim@addtoline
         \expandafter{\the\@temptokena[}%
       \expandafter\verbatim@addtoline
         \expandafter{\@tempc}%
       \def\next{\verbatim@!}%
     \else \expandafter\def\expandafter\@tempc\expandafter
       {\@tempc#1}\fi\fi\fi
     \next}%
\def\verbatim@@testend{%
   \ifx\@tempc\@currenvir
     \verbatim@finish
     \edef\next{\noexpand\end{\@currenvir}%
                \noexpand\verbatim@rescan{\@currenvir}}%
   \else
     \expandafter\verbatim@addtoline
       \expandafter{\the\@temptokena[}%
       \expandafter\verbatim@addtoline
         \expandafter{\@tempc]}%
```

```
          \let\next\verbatim@
        \fi
        \next}%
    \def\verbatim@rescan#1#2~{\if\noexpand~\noexpand#2~\else
        \@warning{Characters dropped after `\string\end{#1}'}\fi}}

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

7 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.

```
#-----------------------------------------------------------------------
# Purpose: generation of the documentation of the struktex package
```

```
# Notice:  this file can be used only with dmake and the option "-B";
#          this option lets dmake interpret the leading spaces as
#          distinguishing characters for commands in the make rules.
#
# Rules:
#          - all-de:     generate all the files and the (basic) german
#                        documentation
#          - all-en:     generate all the files and the (basic) english
#                        documentation
#          - test:       format the examples
#          - history:    generate the documentation with revision
#                        history
#          - develop-de: generate the german documentation with revision
#                        history and source code
#          - develop-en: generate the english documentation with
#                        revision history and source code
#          - realclean
#          - clean
#          - clean-example
#
# Author:  Jobst Hoffmann, Fachhochschule Aachen, Abt. Juelich
# Date:    2003/04/18
#-------------------------------------------------------------------------

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

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

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

# The directory, where to install *.cls and *.sty
CLSDIR=$(INSTALLTEXMF)/tex/latex/jhf/$(PACKAGE)

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

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

```
656
657 # The directory, where to install demo-files
658 # If we have some, we have to add following 2 lines to install rule:
659 #     $(MKDIR) $(DEMODIR); \
660 #     $(INSTALL) $(DEMO_FILES) $(DEMODIR); \
661 DEMODIR=$(DOCDIR)/demo
662
663 # We need this, because the documentation needs the classes and packages
664 # It's not really a good solution, but it's a working solution.
665 TEXINPUTS := $(PWD):$(TEXINPUTS)
666
667 ###########################################################################
668 #   End of customization section
669 ###########################################################################
670
671 DVIPS = dvips
672 LATEX = latex
673 PDFLATEX = pdflatex
674
675 # postscript viewer
676 GV = gv
677
678 COMMON_OPTIONS = \OnlyDescription\CodelineNumbered
679 HISTORY_OPTIONS = \RecordChanges
680 DEVELOPER_OPTIONS = \EnableCrossrefs\RecordChanges\AlsoImplementation\CodelineIndex
681
682 PACKAGE = struktex
683
684 all-de: $(PACKAGE).de.pdf
685
686 all-en: $(PACKAGE).en.pdf
687
688 # strip off the comments from the package
689 $(PACKAGE).sty $(PACKAGE)-test-*.tex: $(PACKAGE).dtx
690
691 $(PACKAGE).sty $(PACKAGE)-test-*.tex: $(PACKAGE).ins
692  +$(LATEX) $<
693
694 # generate the documentation
695 $(PACKAGE).dvi: $(PACKAGE).sty
696
697 $(PACKAGE).de.dvi: $(PACKAGE).dtx
698  +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{$<}"
699  +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{$<}"
700  +mv $(<:.dtx=.dvi) $(<:.dtx=.de.dvi)
701
702 $(PACKAGE).de.pdf: $(PACKAGE).dtx
703  +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{$<}"
704  +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\input{$<}"
705  +mv $(<:.dtx=.pdf) $(<:.dtx=.de.pdf)
706
707 $(PACKAGE).en.dvi: $(PACKAGE).dtx
708  +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\selectlanguageEnglish{}\input{$<}"
709  +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\selectlanguageEnglish{}\input{$<}"
```

```
710  +mv $(<:.dtx=.dvi) $(<:.dtx=.en.dvi)
711
712 $(PACKAGE).en.pdf: $(PACKAGE).dtx
713  +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\selectlanguageEnglish{}\input{$<}"
714  +$(PDFLATEX) "\AtBeginDocument{$(COMMON_OPTIONS)}\def\selectlanguageEnglish{}\input{$<}"
715  +mv $(<:.dtx=.pdf) $(<:.dtx=.en.pdf)
716
717 # generate the documentation with revision history (only german)
718 history: $(PACKAGE).dtx
719  +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(HISTORY_OPTIONS)}\input{$<}"
720  +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(HISTORY_OPTIONS)}\input{$<}"
721  +makeindex -s gind.ist                $(PACKAGE).idx
722  +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
723  +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(HISTORY_OPTIONS)}\input{$<}"
724
725 # generate the documentation for the developer (revision history always
726 # in german)
727 develop-de: $(PACKAGE).dtx
728  +$(LATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)$(DEVELOPER_OPTIONS)}\input{$<}"
729  +$(LATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)$(DEVELOPER_OPTIONS)}\input{$<}"
730  +makeindex -s gind.ist                $(PACKAGE).idx
731  +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
732  +$(LATEX) "\AtBeginDocument{$(HISTORY_OPTIONS)$(DEVELOPER_OPTIONS)}\input{$<}"
733 ifneq (,$(findstring pdf,$(LATEX)))
734  +mv $(<:.dtx=.pdf) $(<:.dtx=.de.pdf)
735 else
736  +mv $(<:.dtx=.dvi) $(<:.dtx=.de.dvi)
737 endif
738
739 develop-en: $(PACKAGE).dtx
740  +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(DEVELOPER_OPTIONS)}\def\selectlanguageEnglish{
741  +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(DEVELOPER_OPTIONS)}\def\selectlanguageEnglish{
742  +makeindex -s gind.ist                $(PACKAGE).idx
743  +makeindex -s gglo.ist -o $(PACKAGE).gls -t $(PACKAGE).glg $(PACKAGE).glo
744  +$(LATEX) "\AtBeginDocument{$(COMMON_OPTIONS)$(DEVELOPER_OPTIONS)}\def\selectlanguageEnglish{
745 ifneq (,$(findstring pdf,$(LATEX)))
746  +mv $(<:.dtx=.pdf) $(<:.dtx=.en.pdf)
747 else
748  +mv $(<:.dtx=.dvi) $(<:.dtx=.en.dvi)
749 endif
750
751 # format the example/test files
752 test:
753  for i in `seq 1 3`; do \
754      f=$(PACKAGE)-test-$$i; \
755      echo file: $$f; \
756      $(LATEX) $$f; \
757      $(DVIPS) -o $$f.ps $$f.dvi; \
758      $(GV) $$f.ps \&; \
759  done
760
761 install: $(PACKAGE).dtx $(PACKAGE).dvi
762  [ -d $(CLSDIR) ] || mkdir -p $(CLSDIR)
763  [ -d $(DOCDIR) ] || mkdir -p $(DOCDIR)
```

```
764 [ -d $(SRCDIR) ] || mkdir -p $(SRCDIR)
765 cp $(PACKAGE).sty      $(CLSDIR)
766 cp $(PACKAGE).dvi      $(DOCDIR)
767 cp $(PACKAGE).ins      $(SRCDIR)
768 cp $(PACKAGE).dtx      $(SRCDIR)
769 cp $(PACKAGE)-test-*.tex   $(SRCDIR)
770 cp LIESMICH        $(SRCDIR)
771 cp README          $(SRCDIR)
772 cp THIS-IS-VERSION-$(VERSION)  $(SRCDIR)
773
774 uninstall:
775 rm -f  $(CLSDIR)/$(PACKAGE).sty
776 rm -fr $(DOCDIR)
777 rm -fr $(SRCDIR)
778
779 pack: $(PACKAGE).de.pdf $(PACKAGE).en.pdf  $(PACKAGE).dtx  $(PACKAGE).ins \
780 LIESMICH README
781 + tar cfvz  $(PACKAGE).tgz $^
782
783 clean:
784 -rm -f *.log *.aux *.brf *.idx *.ilg *.ind
785 -rm -f *.glg *.glo *.gls *.lof *.lot *.out *.toc *.tmp *~
786 -rm *.mk *.makemake
787
788 realclean:  clean
789 -rm -f *.sty *.cls *.ps *.dvi *.pdf
790 -rm -f *test* getversion.* Makefile
791
792 clean-test:
793 rm $(PACKAGE)-test-*.* # this $-sign is needed for font-locking in XEmacs only
```

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.

```
794 sed -e "‘echo \"s/^ /@/g\" | tr ’@’ ’\011’‘" struktex.mk > Makefile
```

Die folgende Datei dient allein dazu, die Version des Paketes zu ermitteln.

```
795 \documentclass[english]{ltxdoc}
796 \nofiles
797 \usepackage{struktex}
798 \GetFileInfo{struktex.sty}
799 \typeout{VERSION \fileversion}
800 \begin{document}
801 \end{document}
```

Literatur

[Fut89] Gerald Futschek. *Programmentwicklung und Verifikation.* Springer Verlag, Wien – New York, 1989. 2

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

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

[Knu86] D. E. Knuth. *The TeX-Book*. Addison-Wesley, Reading, Massachusetts, 1986.

[MDB94] Frank Mittelbach, Denys Duchier and Johannes Braams. *The* DocStrip *program*, Dezember 1994.

[MDB01] Frank Mittelbach, Denys Duchier, Johannes Braams, Marcin Woliński and Mark Wooding *The* DocStrip *program*, September 2001. 3

[Mit94] Frank Mittelbach. *The* doc *and* shortvrb *Packages*, Oktober 1994.

[Mit01] Frank Mittelbach. *The* doc *and* shortvrb *Packages*, September 2001. 3

[Neu96] Marion Neubauer. Feinheiten bei wissenschaftlichen Publikationen – Mikrotypographie-Regeln, Teil I. *Die TeXnische Komödie*, 8(4):23–40, Februar 1996. 26

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