Das moreverb Paket*

Robin Fairbairns (rf10@cam.ac.uk)
nach
Angus Duggan, Rainer Schöpf and Victor Eijkhout†

2011-09-13

Contents

1 Dieses Paket

Dieses Paket nutzt die Ausstattung des verbatim Pakets der LaTeX 2ε-*tools*-Distribution, um eine Anzahl von Dingen zur Verfügung zu stellen, die in der Entwicklung des Pakets als unnötig abgelehnt wurden. (Dennoch gehört der Tabulator-Erweiterungs-Code dieses Pakets zu einem der FAQs von `comp.text.tex`)

Das Paket stellt Dinge dreier breiter Gebiete zur Verfügung:

- Tabulator-Erweiterung und Ähnliches,
- Zeilennummerierung,
- Sonstiges: wortwörtlich in eine Datei schreiben (z. B. für späteres Wiedereinsetzen), und das 'Verpacken'.

*Diese Datei hat die Versionnummer v2.3a, letzte Überarbeitung 2008/06/03
†Übersetzt von **Matthias Biniok** (FSU Jena), September 2011.

1.1 Tabulator-Erweiterung

Das Paket ermöglicht Ihnen, die voraussichtliche Breite der Tabulation festzulegen und erlaubt das Einbinden von Dateien, die Tabulatoren enthalten.

`verbatimtab`

`\begin{verbatimtab}[`⟨*Tab-Breite*⟩`]` gibt dessen Körper[1] wortwörtlich wieder, wobei die Tabulatoren die Größe der gegeben Breite haben (der Standardwert ist 8).

`\verbatimtabinput`

`\verbatimtabinput[`⟨*Tab-Breite*⟩`]{`⟨*Datei-Name*⟩`}` ist die Datei-Einbindungsversion der `verbatimtab`-Umgebung.

`\verbatimtabsize`

Die Größe des Tabulators wird in `\verbatimtabsize` gespeichert und dauert zwischen den Benutzungen der Umgebungen an. (Das heißt, dass ein optionales Argument an einem Element auch alle nachfolgenden Elemente betrifft.)
Um den Wert durch etwas anderes als durch Benutzung eines optionalen Arguments zu ersetzen, schreibt man:

`\renewcommand\verbatimtabsize{`⟨*Wert*⟩`\relax}`

Es gibt keine Versprechungen hinsichtlich der gebotenen Leistung, wenn Sie das `\relax` weglassen!

1.2 Zeilennummerierung

Zeilennummerierung ist oft bei der Abbildung von Code-Beispielen nützlich (dies ist nötig für diejenigen von uns, die solche Code-Schnipsel nicht verschönigen wollen).

`listing`

`\begin{listing}[`⟨*Intervall*⟩`]{`⟨*Startzeile*⟩`}` nummeriert die Zeilen des Körpers. Das Argument ⟨*Startzeile*⟩ gibt die Startzeile an. Das optionale Argument ⟨*Intervall*⟩ legt die Anzahl der Zeilen zwischen den nummerierten Zeilen fest: Das heißt, jede Zeile mit der Nummer $= 0 \pmod{\langle Intervall \rangle}$ wird nummeriert. (Außerdem wird Zeilennummer 1 immer nummeriert.) Der Standardwert von ⟨*Intervall*⟩ ist 1 (d. h., es wird jede Zeile nummeriert).

`listingcont`

`\begin{listingcont}` fährt dort fort, wo der letzte Eintrag aufgehört hat.

Der Stil, in dem das Label gesetzt ist, kann für eine der beiden Umgebungen verändert werden, durch Neudefinierung von `\listinglabel`. Beide Umgebungen erweitern auch Tabulatoren.

'`*`'-Versionen von beiden Listing-Umgebungen werden angeboten; diese machen die gewöhnlichen `verbatim*` Dinge mit ausgegebenen Leerzeichen wie '␣', aber erweitern keine Tabulatoren.

`listinginput`

`\listinginput[`⟨*Intervall*⟩`]{`⟨*Startzeile*⟩`}{`⟨*Dateiname*⟩`}` ist eine Listing-Version zum Schreiben in eine Datei. Es gibt keine '`*`'-Form.

[1] Mit Körper ist immer der Rumpf des Befehls gemeint.

1.3 Sonstiges

verbatimwrite

`\begin{verbatimwrite}{`⟨*Dateiname*⟩`}` schreibt allen Text seines Körpers in eine Datei, dessen Name als Argument übergeben wird.

boxedverbatim

`\begin{boxedverbatim}` legt den Inhalt einer verbatim-Umgebung in einer Rahmenbox ab. Wenn man versucht, dies auf eine 'naive' Weise zu tun, wird man sehen, dass die verbatim-Zeilen alle so breit werden, wie die Breite der Seite, sodass die Box häufig sehr schlecht zum Text passt, den sie umgibt.

verbatimcmd

Die `verbatimcmd` Umgebung wurde von der LaTeX2.09 und früheren LaTeX 2_ε Versionen dieses Paketes angeboten. Jedoch werden dessen Fähigkeiten nun von `alltt` zur Verfügung gestellt, das im alltt-Paket definiert ist, welches jetzt Teil der LaTeX Grunddistribution ist, daher wurde `verbatimcmd` zurückgezogen.

2 Der Code des Pakets

1 ⟨∗moreverb⟩

2.1 Initial-Code

Lädt das verbatim-Paket, falls es noch nicht geladen ist.

```
2 \@ifundefined{verbatim@processline}{\RequirePackage{verbatim}}{}
```

2.2 In eine Datei schreiben

verbatimwrite

`\begin{verbatimwrite}{`⟨*Dateiname*⟩`}` schreibt den gesamten Text im Körper in eine Datei, der Name der Datei wird als Argument übergeben. (Dieser Code wurde von Rainer Schöpf geschrieben.)

Beachten Sie, dass der Code bei erstmaliger Benutzung seinen eigenen Output-Stream erzeugt. (Dies ist eine Sparmaßnahme; wenn der Benutzer `\begin{verbatimwrite}` nie benutzt, wird auch kein `\write`-Stream erzeugt. Ein gegenwärtiger problematischer Benutzungsfall taucht in tex.sx auf . . .)

```
3 \def\verbatimwrite#1{%
4   \@ifundefined{verbatim@out}{\newwrite\verbatim@out}{}%
5   \@bsphack
6   \immediate\openout \verbatim@out #1
7   \let\do\@makeother\dospecials
8   \catcode`\^^M\active \catcode`\^^I=12
9   \def\verbatim@processline{%
10    \immediate\write\verbatim@out
11      {\the\verbatim@line}}%
12  \verbatim@start}
13 \def\endverbatimwrite{%
14  \immediate\closeout\verbatim@out
15  \@esphack}
```

2.3 Tabulator-Erweiterung

Wir definieren ein paar zusätzliche Makros und Zähler für erweiternde Tabulatoren. Sie werden von den `listing`- und `verbatimtab`-Umgebungen verwendet.

```
\newcount\tab@position \newcount\tab@size
```

`\verbatimtabsize` ist gewöhnlich ein Zähler, doch das scheint mir übertrieben (LaTeX benutzt einfach zu viele Zähler...).

```
\def\verbatimtabsize{8\relax}
```

`\@xobeytab` `\@xobeytab` fügt genug Leerzeichen ein, um uns zum nächsten symbolischen Tabulatoren-Ende zu bringen.

```
\def\@xobeytab{%
  \loop
    \toks@\expandafter{\the\toks@\@xobeysp}%
    \advance\tab@position-1
  \ifnum\tab@position>0 \repeat
}
```

`\@vobeytabs` `\@vobeytabs` initialisiert die Benutzung von `\@xobeytab`. Muss innerhalb einer Gruppe ausgeführt werden, da es nicht in die weite Welt entweichen darf.

```
\begingroup
  \catcode`\^^I=\active
  \gdef\@vobeytabs{\catcode`\^^I\active\let^^I\@xobeytab}%
\endgroup
```

`\verbatim@tabexpand` `\verbatim@tabexpand`⟨*body of line*⟩`\@nil` verarbeitet jeden Buchstaben einer Zeile durch Endrekursion, zählt die Buchstaben und jongliert ein bisschen, wenn ein Tab auftritt. (Was gewöhnlich 'line imaging' genannt wird...)

```
\def\verbatim@tabexpand#1{%
  \ifx#1\@nil
%   \showthe\toks@
    \the\toks@
    \expandafter\par
  \else
    \ifx#1\@xobeytab
      \@xobeytab
    \else
```

Wir können `\@xobeysp` gefahrlos in das Token-Register tun, da es genau das macht, was wir brauchen.

```
      \toks@\expandafter{\the\toks@#1}%
      \advance\tab@position\m@ne
    \fi
    \ifnum\tab@position=0 \tab@position\tab@size \fi
    \expandafter\verbatim@tabexpand
  \fi
}
```

listing `\begin{listing}[`⟨*Intervall*⟩`]{`⟨*Startzeile*⟩`}`

Definiert eine verbatim-Umgebung mit nummerierten Zeilen; das optionale Argument ⟨*Intervall*⟩ legt die Anzahl der Zeilen zwischen den nummerierten Zeilen fest, und das Argument ⟨*Startzeile*⟩ gibt die Startzeile an.

listingcont `\begin{listingcont}`

Fährt dort fort, wo die Auflistung aufhört. Der Stil, in dem das Label gesetzt ist, kann durch Neudefinierung von `\listinglabel` verändert werden.

'`*`'-Versionen beider Umgebungen werden angeboten.

`\listing@line` `\listing@line` enthält die aktuelle Zeilennummer; der Standardwert ist 1, also kann man `listingcont` fröhlich in einem Dokument benutzen, wenn nur ein Stream von wörtlichem Text geschrieben wird.

```
\newcount\listing@line \listing@line=1
```

`\listing@step` `\listing@step` ist ein anderer Fall, bei dem ein Zähler gewohnt verwendet wird, ohne von sehr offensichtlichem Nutzen zu sein, aber ein wertvolles Zählerregister aufbraucht. Auch hier ist der Wert modal; das hintere `\relax` ist notwendig.

```
\def\listing@step{1\relax}
```

Das Hinzufügen einer `\hbox` vor der Zeile verursacht einen Zeilenumbruch, also mache ich schnell dieses Geschwätz,[2] um die Linien schön ausgerichtet zu bekommen. Ich habe wahrscheinlich einige offensichtliche Gründe vergessen, weshalb `\hbox`es nicht funktionieren. [3].

```
\def\listinglabel#1{\llap{\small\rmfamily\the#1}\hskip\listingoffset\relax}
\def\thelisting@line{%
  \setbox0\hbox{\listinglabel\listing@line}%
  \@tempcnta=\listing@line
  \divide\@tempcnta\listing@step \multiply\@tempcnta\listing@step
  \ifnum\listing@line=\@ne
    \unhbox0
  \else
    \ifnum\@tempcnta=\listing@line
      \unhbox0
    \else
      \hskip\wd0
    \fi
  \fi}
```

`\listingoffset` `\listingoffset` ist die Trennung zwischen der Zeilennummer und der aktuellen Zeile; der Standardwert ist `1.5em`.

```
\providecommand\listingoffset{1.5em}
```

[2] Das Personalpronomen war in den Kommentaren der Originalversion dieses Pakets vorhanden; ich weiß nicht, auf wen es sich bezieht — RF

[3] Der Grund ist, dass `\hbox` im vertikalen Modus einen kompletten Paragraphen aus eigener Kraft herstellt; dieses Problem könnte im Laufe der Zeit behandelt werden, aber momentan...

Benutzen Sie einfach `\listing`, um die Parameter einzulesen und um dann `\listingcont` zu benutzen.

```
\newcommand\listing[2][1]{%
  \global\listing@line=#2\relax
  \gdef\listing@step{#1\relax}
  \listingcont}
```

`\listingcont` ist das funktionale Ende der zwei Umgebungen.

```
\def\listingcont{%
  \tab@size=\verbatimtabsize
  \def\verbatim@processline{\tab@position\tab@size
    \thelisting@line \global\advance\listing@line1
    \toks@{}%
    \expandafter\verbatim@tabexpand\the\verbatim@line\@nil}%
  \@verbatim\frenchspacing\@vobeyspaces\@vobeytabs\verbatim@start}
```

Nichts Besonderes am Ende der zwei Umgebungen.

```
\let\endlisting=\endtrivlist
\let\endlistingcont=\endtrivlist
```

Jetzt dasselbe Geschwätz für die '`*`' Versionen.

```
\expandafter\newcommand\csname listing*\endcsname[2][1]{%
  \global\listing@line=#2\relax
  \gdef\listing@step{#1\relax}
  \csname listingcont*\endcsname}
\@namedef{listingcont*}{%
  \def\verbatim@processline{%
    \thelisting@line \global\advance\listing@line1
    \the\verbatim@line\par}%
  \@verbatim\verbatim@start}
```

Nur ein bisschen Ärger in den Namensdefinitionen am Ende der Umgebungen.

```
\expandafter\let\csname endlisting*\endcsname\endtrivlist
\expandafter\let\csname endlistingcont*\endcsname\endtrivlist
```

`listinginput` `\listinginput[`⟨*Intervall*⟩`]{`⟨*Startzeile*⟩`}{`⟨*Dateiname*⟩`}` ist eine Version von listing zum Schreiben in eine Datei.

```
\def\listinginput{%
   \@ifnextchar[%]
     {\@listinginput}%
     {\@listinginput[1]}}
\begingroup
  \catcode`\~=\active \lccode`\~=`\^^M \lccode`\N=`\N
  \lowercase{\endgroup
    \def\@listinginput[#1]#2#3{\begingroup
      \global\listing@line=#2
      \gdef\listing@step{#1\relax}
      \tab@size=\verbatimtabsize
      \def\verbatim@processline{\tab@position\tab@size
```

```
        \thelisting@line \global\advance\listing@line1
        \toks@{}%
        \expandafter\verbatim@tabexpand\the\verbatim@line\@nil}%
      \@verbatim\frenchspacing\@vobeyspaces\@vobeytabs
      \def\verbatim@addtoline##1~{%
        \verbatim@line\expandafter{\the\verbatim@line##1}}%
      \openin\verbatim@in@stream=#3
      \ifeof\verbatim@in@stream
        \PackageWarning{moreverb}{No file #3.}%
      \else
        \do@verbatimtabinput
        \closein\verbatim@in@stream
      \fi
      \endtrivlist\endgroup
    \@doendpe
  }%
}
```

`verbatimcmd` `verbatimcmd` war eine verbatim-Umgebung mit Ausnahme der Escape- und Gruppierungselemente `\`, `{`, `}`. Das ist (err) genau die Spezifikation der `alltt` Umgebung, und es ist im alltt Paket, welches jetzt Teil der Grunddistribution ist.

```
\def\verbatimcmd{%
  \PackageError{moreverb}{The verbatimcmd environment is obsolete}%
                         {Use alltt (from the LaTeX required package
                           alltt) in place of verbatimcmd}%
}
\let\endverbatimcmd\relax
```

`boxedverbatim` `boxedverbatim` legt den Inhalt einer verbatim-Umgebung in einer Rahmenbox ab.
(Geschrieben von Victor Eijkhout.)
Fehlerbehebung:

- David Carlisle 1995-12-28, Beschäftigung mit Platzthemen (iirc)
- Moretn Høgholm 2008-06-01, Positionierung von Rahmen in Listen

Zuerst redefiniert man 'processline', um nur eine Zeile, so breit wie die natürliche Breite der Zeile, herzustellen.

```
\def\boxedverbatim{%
  \def\verbatim@processline{%
    {\setbox0=\hbox{\the\verbatim@line}%
    \hsize=\wd0 \the\verbatim@line\par}}%
```

Nun speichert man den verbatim-Code in einer Box.

```
  \@minipagetrue           % DPC
  \@tempswatrue            % DPC
  \@totalleftmargin\z@     % MH
  \setbox0=\vbox\bgroup \verbatim
}
```

Am Ende der Umgebung, müssen wir (umm) die Resultate einfach in einen Rahmen stecken.

```
\def\endboxedverbatim{%
  \endverbatim
  \unskip\setbox0=\lastbox % DPC
```

Jetzt ist alles in der Box, also können wir sie schließen...

```
  \egroup
```

Um den Code zum Zentrieren zu ändern, benötigt die nächste Zeile eine Stelle zum Schnippeln.

```
  \fbox{\box0}%
}
```

verbatimtab `\begin{verbatimtab}[`⟨*tab width*⟩`]` ist eine verbatim-Umgebung, welche Tabulatoren erweitert; das optionale Argument legt den Abstand zwischen den Tab-Stops fest.

Das Ausführen von `\obeylines`, bevor man sich das optionale Argument ansieht, verhindert eine leere erste Zeile der Umgebung, die ein `\par` Zeichen wird (dieser Fehler wurde von Werner Lemberg berichtet).

```
\newenvironment{verbatimtab}{\obeylines\@verbatimtab}{\endtrivlist}
```

Dies verarbeitet das optionale Argument von `verbatimtab`, sodass wir uns nun von den gefürchteten `\par` Zeichen geschützt haben:

```
\newcommand\@verbatimtab[1][\verbatimtabsize]{%
  \do@verbatimtab{#1}{%
     \@verbatim\frenchspacing\@vobeyspaces\@vobeytabs\verbatim@start}%
}
```

\do@verbatimtab Stellt eine Umgebung für Tabulatoren bereit; `#1` ist der Wert der Tabulator-Größe (in der Regel, ursprünglich ein optionales Argument), `#2` sind die 'Start-Kommandos', die ausgeführt werden sollen, sobald eine passende Definition von `\verbatim@processline` eingeführt wurde:

```
\def\do@verbatimtab#1#2{%
  \tab@size=#1
  \def\verbatim@processline{\tab@position\tab@size
    \toks@{}%
    \expandafter\verbatim@tabexpand\the\verbatim@line\@nil}%
  #2%
}
```

\verbatimtabinput `\verbatimtabinput[`⟨*Tab-Breite*⟩`]{`⟨*Dateiname*⟩`}` ist die Version der `verbatimtab`-Umgebung zum Schreiben in eine Datei.

Wir benutzen den Input-Stream der vom verbatim-Paket erworben wurde; wir benötigen es schlielich zum Laden. (Man muss zugeben, dass der Name des Streams nicht wirklich Teil der definierten Schnittstelle des Pakets ist, aber auf der anderen Seite gibt es keine besondere Wahrscheinlichkeit, dass es sich jemals ändern wird.)

Wir benutzen (ursprünglich) keine ausgefallenen Eigenschaften von `\newcommand`, da die Definition innerhalb einer Gruppe war, und damit global. Also ... 'traditioneller' Code, um einen Befehl mit einem optionalen Argument anzubieten (was wahrscheinlich nicht mehr nötig ist):

```
\def\verbatimtabinput{%
  \@ifnextchar[%]
    {\@verbatimtabinput}%
    {\@verbatimtabinput[\verbatimtabsize]}}
\begingroup
\catcode`\~=\active \lccode`\~=`\^^M \lccode`\N=`\N
\lowercase{\endgroup
  \def\@verbatimtabinput[#1]#2{\begingroup
    \do@verbatimtab{#1}{%
      \@verbatim\frenchspacing\@vobeyspaces\@vobeytabs}%
    \def\verbatim@addtoline##1~{%
      \verbatim@line\expandafter{\the\verbatim@line##1}}%
    \openin\verbatim@in@stream=#2
    \ifeof\verbatim@in@stream
      \PackageWarning{moreverb}{No file #2.}
    \else
      \@addtofilelist{#2}%
      \do@verbatimtabinput
      \closein\verbatim@in@stream
    \fi
  \endtrivlist\endgroup\@doendpe}%
}
```

`\do@verbatimtabinput` Ausgeschriebene Schleife (Endrekursion) zum Lesen der Datei:

```
\def\do@verbatimtabinput{%
  \read\verbatim@in@stream to \verbtab@line
  \ifeof\verbatim@in@stream
  \else
    \expandafter\verbatim@addtoline\verbtab@line
    \verbatim@processline
    \verbatim@startline
    \expandafter\do@verbatimtabinput
  \fi
}
```

⟨/moreverb⟩