Zum TUG-DD-Treffen am 14.11.'07 angesprochene Themen

Tobias Nähring

17. November 2007

Inhaltsverzeichnis

1 Einleitung

Der Vortrag von Gerrit zu `tikz` wurde auf den voraussichtlichen Termin 19.12.'07 verschoben. Ersatzweise haben wir die Zeit für die Auswertung einiger Fragen vom Linux-Info-Tag'07 und zur Bearbeitung einiger eigener Problemstellungen genutzt. Ich habe auch eine ganze Weile aus dem TEX-Nähkästchen geplaudert. Das spiegelt sich hier in dem längsten Abschnitt 4 wider. Im Nachgang habe ich den Text zu den angesprochenen Themen noch etwas ergänzt.

Viel Spaß beim Lesen,
Tobias Nähring

2 Mehrspaltige align-Umgebung

Die `align`-Umgebung kann auch mehrspaltig genutzt werden. An jedem zweiten Ausrichtungszeichen `&` beginnt eine neue Spalte. An den dazwischen liegenden Ausrichtungszeichen werden die links und rechts davon stehenden Zeichenketten jeweils rechts bzw. links ausgerichtet. Die LaTeX-Zeilen

```
\begin{align}
  E&=mc^{2} & x'&=\frac{x-vt}{\sqrt{1-\beta^{2}}}\\
  z^{2}&=x^{2}+y^{2} & m'&=\frac{m}{\sqrt{1-\beta^{2}}}\\
  x&=z\cos(\phi)
\end{align}
```

liefern also:

$$E = mc^2 \qquad x' = \frac{x - vt}{\sqrt{1 - \beta^2}} \tag{1}$$

$$z^2 = x^2 + y^2 \qquad m' = \frac{m}{\sqrt{1 - \beta^2}} \tag{2}$$

$$x = z\cos(\phi) \tag{3}$$

3 `Emacs` mit `preview` und `reftex`

Ich (Tobias) habe die Elisp-Pakete `preview` und `reftex` für den `emacs` vorgestellt, die das Editieren von LaTeX-Texten vereinfachen.

`Preview` ersetzt auf Nutzerwunsch hin innerhalb des Eingabepuffers LaTeX-Fragmente, die stark von ihrem natürlichen Erscheinungsbild abweichen, durch LaTeX-generierte Bilder. Die Bilder sehen schon so aus, wie sie später im Dokument erscheinen. Auch die Nummerierung (z.B. Gleichungsnummerierung) stimmt schon. Zu den übersetzten LaTeX-Fragmenten zählen

- Gleichungen (eingebettete und abgesetzte),
- Bilder (`figure`-Umgebungen)
- Überschriften

`RefTeX` stellt Listen aller im Text auftretenden nummerierten Formeln, Abschnitte u.s.w. zur Verfügung. Aus diesen Listen kann man einen gewünschten Eintrag per Cursor auswählen (auch bei mehrzeiligen Einträgen nur ein Tastendruck pro Eintrag) und `reftex` generiert selbstständig einen Verweis im LaTeX-Text. Die im aktuellen Dokument eingebundenen BibTeX-Dateien können bequem nach einem regulären Ausdruck (z.B. einem Autornamen) durchsucht werden. Daraufhin wird eine Trefferliste von Literaturstellen angezeigt, aus der man wieder einen passenden Eintrag auswählen kann, für den dann im LaTeX-Dokument eine Referenz erstellt wird.

In den generierten Listen wird immer ein kleines Stück Kontext mit angezeigt, so dass man zum Beispiel meistens die relevante Formel oder den Buchtitel und den Autor erkennt.

Bei Debian ist das `reftex`-Paket mit im übergeordneten `auctex`-Paket enthalten, das Paket `preview-latex` muss extra installiert werden.

4 TEX-Spielereien

Auf einem früheren Treffen haben wir schon einmal kurz die Methode diskutiert, mit der im Beispiel aus Abschnitt 2 innerhalb der Verbatim-Umgebung einzelne Zeichen rot hervorgehoben werden. Am 14.11.'07 sahen wir uns die TEXnischen Grundlagen dafür noch einmal genauer an. Die „Features“, die hier für die Verbatim-Umgebung schrittweise aufgebaut werden, gibt es natürlich schon in tausendmal besserer Version fixundfertig in `fancyvrb` verpackt. Hier geht es also mehr um das TEXnische Verständnis...

4.1 Catcodes

Dass das Eingabezeichen '\' TEX-Befehlssequenzen und Makrodefinitionen einleitet, '%' einen Zeilenkommentar einleitet, '$' eingebettete Formeln einschließt, und so weiter, das ist alles nicht fest in TEX eingemeiselt. Vielmehr bekommen diese Eingabezeichen ihre spezielle Bedeutung erst dadurch, dass ihnen ein sogenannter *Kategoriecode* (TEX: *catcode*) zugeordnet wird.

Zum Beispiel ist den ASCII-Eingabezeichen 'A',...,'Z','a',...,'z' beim Start des TEX-Interpreters der Catcode 11 zugewiesen, den D. E. Knuth als *letter* bezeichnet. Zeichenketten dieser Kategorie werden (normalerweise) unter Berücksichtigung von Ligaturen und Trennmuster direkt Ausgabezeichen zugeordnet.

Die Ziffern '0',...,'9' gehören zur Kategorie 12, der D. E. Knuth den Namen *other* verpasst hat. Den Zeichen dieser Kategorie werden (normalerweise) auch direkt Ausgabezeichen verpasst, wobei jedoch vom TEX-Interpreter nicht so stark auf Wechselwirkungen aufeinanderfolgender Zeichen geachtet werden muss.

Das Zeichen '\' hat beim Start des TEX-Interpreters den Catcode 0. Durch diesen Catcode wird '\' zum TEX-Escape-Zeichen: Eine darauf folgende Zeichenkette aus letter-Zeichen wird von TEX als Name einer Kommandosequenz interpretiert.

In Tabelle 1 habe ich noch einmal alle TEX-Catcodes zusammen mit ihrer Bedeutung aufgelistet. Für das Abfragen und Setzen des Catcode eines Eingabezeichens gibt es das TEX-Kommando `\catcode`.

Zum Beispiel liefert die Zeile

```
\the\catcode`\@
```

die Ausgabe 12. Mit `` `\ `` wird das ASCII-Zeichen '@' in einen zugehörigen TEX-internen Zeichencode übersetzt. Das Kommando `\catcode` liefert den zu diesem

Catcode	Standard	Bedeutung
0	\	TEX-Escape-Zeichen: mit einem Zeichen dieser Kategorie werden Kommandosequenzen, (also TEX-Befehle und Makros) eingeleitet.
1	{	begin-group: Anfang einer Gruppe, Gruppen begrenzen Makroargumente aus mehreren Zeichen, legen eine lokale Definitionsebene fest, begrenzen bei einer Definition den zu expandierenden Text
2	}	end-group: Ende einer Gruppe, s.o.
3	$	math-shift, wechselt in den Mathe-Modus und zurück
4	&	alignment-tab: Ausrichtungszeichen in Tabellen, abgesetzten Formeln, Matrizen u.s.w.
5	endline	endline-character: begrenzt Zeilenkommentare, wirkt ansonsten wie Blank Space
6	#	macro parameter character: mit dem Zeichen dieser Kategorie und einer Zahl 0...9 werden innerhalb von Makrodefinitionen die Makroargumente bezeichnet.
7	^	superscript
8	_	subscript
9		ignored: Zeichen dieser Kategorie werden ignoriert
10	␣	blank space: Leerzeichen
11	a,...,Z	letter: diesen Eingabezeichen werden direkt Ausgabezeichen zugeordnet; außerdem können hiermit auch längere Namen für Kommandosequenzen gebildet werden (wie z.B. `\documentclass`).
12	0,...,9	other: stehen als Einzelzeichen, wie z.B. Zahlen, Kommandosequenznamen können nur aus einem solchen Zeichen bestehen (wie z.B. \-)
13	~	active: Zeichen dieser Kategorie können selber als Kommandosequenzen definiert werden (wie z.B. auch " wenn `ngerman` geladen ist)
14	%	comment: Einleitung eines Zeilenkommentars
15		invalid: Muss TEX Zeichen dieser Kategorie schlucken meckert es mit einer Fehlermeldung

Tabelle 1: Kurz erläuterte Kategoriecodes von TEX

Zeichencode gehörigen Catcode 12 als Zahl und `\the` setzt für diese Zahl die Ausgabezeichen ‘1’ und ‘2’.

Mit der Zeile

```
\catcode‘\@=0
```

weist man dem Zeichen ‘`@`’ den Catcode 0 zu, was dieses in ein TEX-Escape-Zeichen umgewandelt. Danach kann man ‘`@`’ anstelle von ‘\’ für LATEX-Kommandos nutzen. Zum Beispiel liefert

```
{
\catcode‘\@=0
@red Dieser Text ist rot @black und dieser schwarz.
}
```

die Ausgabe: Dieser Text ist rot und dieser schwarz.

4.2 Verbatim

Das Makro `\verbatim`, das von LATEX bei `\begin{verbatim}` aufgerufen wird, beraubt alle Zeichen wie ‘\’, ‘$’ u.s.w. ihrer speziellen Bedeutung, indem es die zugehörigen Catcodes auf 12 (=other) setzt. TEX ordnet diesen Eingabezeichen also innerhalb von `\begin{verbatim}...\end{verbatim}` direkt die entsprechenden Ausgabezeichen zu.

Zum Glück erwischt es dabei nicht das ‘`@`’-Zeichen, das ja eigentlich sowieso schon den Catcode 12 hat. Catcodespielchen mit `@` vor einer Verbatim-Umgebung wirken sich also in ihr aus.

4.2.1 Aktive Zeichen in Verbatim

Möchten wir im Verbatim-Text nur einzelne Zeichen rot markieren, wie das im Abschnitt 2 der Fall ist, so können wir vorteilhaft Catcode 13 (=active) nutzen. Einem Eingabezeichen der Kategorie ‘active’ kann man direkt eine Kommandosequenz zuordnen. Man benötigt kein vorgestelltes Escape-Zeichen wie `\` und auch keinen zusätzlichen Kommandonamen. Zum Beispiel liefern die Zeilen

```
{
\catcode‘\@=13
\def@#1{{\red#1}}
@x
}
```

oder gleichwertig

```
{
\catcode‘\@=13
\newcommand@[1]{{\red#1}}
@x
}
```

die Ausgabe x. Im Zusammenhang mit einer Verbatimumgebung kann man dann zum Beispiel

```
{
\catcode`\@=13
\def@#1{{\red#1}}
\begin{verbatim}
\begin{align}
  z^2@&=x^2+y^2\\
  x@&=z\cos(\phi)
\end{align}
\end{verbatim}
}
```

schreiben und erhält die Ausgabe:

```
\begin{align}
  z^2&=x^2+y^2\\
  x&=z\cos(\phi)
\end{align}
```

4.2.2 Makros ohne Argumente in Verbatim

Eventuell benötigen wir im Verbatim-Text benannte Makros, wie z.B. `\red`, `\blue` und `\black`. In diesem Fall wandelt man '`@`' lieber in ein TEX-Escape-Zeichen um und nutzt dieses dann innerhalb der Verbatim-Umgebung zur Eingabe von Befehlssequenzen. Die Zeilen

```
{
\catcode`\@=0
\let@@\relax
\begin{verbatim}
\red=@red@@Rot@black \blue=@blue@@Blau
\end{verbatim}
}
```

liefern beispielsweise die Ausgabe:

```
\red=Rot \blue=Blau
```

Mit `\let@@\relax` haben wir dabei dem Kommando `@@` das interne TEX-Kommando `\relax` zugewiesen. Dieses sagt einfach, dass sich der TEX-Interpreter einmal ausruhen darf und garnichts machen braucht. `@@` wird dadurch zum syntaktischen Hilfsmittel. Warum wir so einen Schnickschnack benötigen? Nunja, innerhalb von `\verbatim` haben die Leerzeichen nicht mehr den Catcode 10 (=blank char), sondern 13 (=active). Das bedeutet, dass sie hinter Kommandonamen nicht mehr verschluckt werden, sondern in der Ausgabe landen. Hätten wir `@@` nicht, so müssten wir hinter `@red` ein Leerzeichen einfügen, um `@red` von `Rot` abzutrennen. Wir müssten also `@red Rot` schreiben. Das Leerzeichen würde dann jedoch dummer Weise mit in der Ausgabe landen.

4.2.3 Makros mit längeren Argumenten in Verbatim

Um auch TEX-Makros mit längeren Argumenten, wie zum Beispiel `\emph{Argumenttest}`, innerhalb von Verbatim-Umgebungen verwenden zu können, ohne dabei neue Zeichen für öffnende und schließende Klammern reservieren zu müssen, nutzen wir eine Parser-Eigenschaft von TEX.

Definiert man ein Kommando in der Form

```
\def\Betonung#1{\emph{#1}}
```

so wird, falls hinter dem Kommando das nächst Nicht-'Blank-Space'-Zeichen der Kategorie 'begin group' liegt, als Argument eine ganze Gruppe eingelesen, ansonsten nur das nächste Nicht-'Blank-Space' Zeichen. Zum Beispiel liefert

```
\Betonung Eins \Betonung{Zwei}
```

die Ausgabe *E*ins *Zwei*.

Man kann bei der Definition jedoch auch andere Muster nutzen, wie in dem Beispiel

```
\def\Kommando#1\\#2\\{#1{#2}}
```

Alles was hinter `\def` und vor dem Zeichen '`{`' der Kategorie 1 (=begin group) steht, möchte TEX einparsen, wenn es `\Kommando` im Eingabestrom sieht. Die Sequenzen `#1` und `#2` vertreten dabei die Zeichenketten für die Argumente des Makros.

Das oben definierte Kommando kann man zum Beispiel in der Form

```
\Kommando\emph\\Argumenttest\\
```

aufrufen. Die zugehörige Ausgabe ist dann: *Argumenttest* Man benötigt beim Aufruf dieses Kommandos mit einem Zeichenkettenargument nur noch *eine* Art spezieller Zeichen. Das brauchen wir jetzt nur noch auf die Verbatim-Umgebung anzuwenden und sind fertig:

```
{
\catcode`\@=0
\def@Kommando#1@@#2@@{#1{#2}}
\begin{verbatim}
In diesem Verbatim-Text nutzen wir \Kommando mit
einem l"angeren @Kommando@emph@@Argument@@.
\end{verbatim}
}
```

Die zugehörige Ausgabe ist:

```
In diesem Verbatim-Text nutzen wir \Kommando mit
einem l"angeren Argument.
```

4.2.4 Das TEX-Escape-Zeichen in Verbatim

Was machen wir, wenn wir das Zeichen, das wir uns als TEX-Escape-Zeichen ausgesucht haben, doch einmal innerhalb einer Verbatim-Umgebung benötigen? Nichts einfacher als das! Wir defnieren uns vor unseren Catcode-Spielereien ein Makro `\realAt`, das dieses Ausgabezeichen erzeugt. Das Makro können wir dann innerhalb der Verbatim-Umgebung benutzen.

```
{
\def\realAt{{\texttt{@}}}
\catcode`\@=0
\begin{verbatim}
Das @realAt-Zeichen innerhalb einer Verbatim-Umgebung.
\end{verbatim}
}
```

Die zugehörige Ausgabe ist:

```
Das @-Zeichen innerhalb einer Verbatim-Umgebung.
```

Dabei nutzen wir aus, dass bereits beim Parsen der Definition der Definitionstext in Token gewandelt wird, die das Verhalten seiner einzelnen Bestandteile festlegen. Es wird also bereits zur Zeit der Definition von `\realAt` festgelegt, das dass Zeichen `@` Catcode 12 hat und somit in der Ausgabe landen soll, und es wird **nicht** etwa erst bei der Expansion von `@realAt` festgestellt, dass `@` ein TEX-Escape-Zeichen ist.

```
F"ur die \TeX-Puristen gibt es auch noch die Variante
\chardef\realAt`\@
die in dieser Verbatimumgebung angewandt wurde, zu der ich aber jetzt
hier nix weiter sage.
(Bei Bedarf k"onnen die Quellen dieses Dokuments inspiziert werden.)
```

4.3 Zentrieren einer Verbatim-Umgebung

Bei unserem Treffen, haben wir auch das Zentrieren von Verbatim-Blöcken angesprochen. Natürlich geht das alles mit der `BVerbatim`-Umgebung von `fancyvrb`. Jedoch wollen wir ja so nebenbei ein wenig TEXeln.

Beim Treffen sind wir irgendwie darauf gekommen, dass sich Verbatim-Umgebungen nicht mit Hilfe einer `minipage` zentrieren ließe. Das geht aber. Diese Variante ist sogar etwas übersichtlicher als die beim Treffen besprochene. Deshalb diskutiere ich hier die `minipage`-Variante.

Die längste Zeile in der Verbatim-Umgebung stopfen wir in eine `\hbox` und geben dann als Breitenparameter der `minipage`-Umgebung die Weite dieser `\hbox` an.

```
\begin{center}
\newbox\mybox
\setbox\mybox\hbox{\verb=Das ist die l"angste Zeile.=}
```

```
\begin{minipage}{\wd\mybox}
\begin{verbatim}
Erste Zeile
Das ist die l"angste Zeile.
Eine Leerzeile:

Letzte Zeile.
\end{verbatim}
\end{minipage}
\end{center}
```

Das ganze sieht dann so aus:

```
Erste Zeile
Das ist die l"angste Zeile.
Eine Leerzeile:

Letzte Zeile.
```

Die Markierung zeigt zum Vergleich die Seitenmitte.

4.4 TEX-Makros mit beliebig beliebig vielen Argumenten

Wir haben uns bereits in Abschnitt 4.2.2 von der Nützlichkeit des TEX-Kommandos für das Nichtstun `\relax` überzeugen können. Beim Treffen habe ich demonstriert, dass man `\relax` gut als Abschluss von Makros mit beliebig vielen Argumenten nutzen kann.

Solche Makros sind meist rekursiv aufgebaut. Am Ende des Definitionstextes expandiert das Makro sich selbst, solange eine vorgegebene Abbruchsbedingung nicht erfüllt ist. Abbrechen kann man die Rekursion zum Beispiel, wenn als Makroargument `\relax` gelesen wird.

In TEX sieht das ungefähr so aus:

```
\def\Serie#1{%
\def\tmp{#1}%
\if\tmp\relax%
  \let\next\relax%
\else%
  \fbox{#1}%
  \let\next\Serie%
\fi%
\next}
```

Bei der Abarbeitung des Makros schaut sich der TEX-Interpreter zuerst einmal das Makroargument an: Das Argument wird in ein Hilfsmakro `\tmp` gestopft (`\def\tmp{#1}`) und dieses Makro wird dann durch `\if\tmp\relax` mit `\relax`

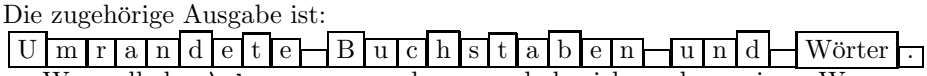
verglichen. Am Ende des Makros wird `\next` expandiert. Solange die Abbruchsbedingung noch nicht erfüllt ist (`\else`-Zweig), stellen wir irgend etwas Sinnvolles mit dem aktuellen Argument an (hier `\fbox{#1}`) und setzen `\next` dem Makro selber gleich (`\let\next\Serie`), wodurch es nach der Expansion am Ende des expandierten Textes selber wieder dasteht. Wurde dagegen `\relax` als Argument gelesen, so wird der `\if`-Zweig durchlaufen und `\next` gleich `\relax` gesetzt. Am Ende des expandierten Textes (`\next`) kann sich TEX in diesem Fall also einfach mal einen Zyklus lang ausruhen, und die Rekursion wird beendet.

Zum `\if`-Kommando bleibt zu sagen, dass es seine zwei Argumente so lange expandiert, bis nur noch reine TEX-Tokensequenzen dastehen, und diese Tokensequenzen werden dann auf Gleichheit getestet (zu den reinen TEX-Kommandos zählt auch `\relax`).

Die folgenden Zeilen demonstrieren eine Anwendung des oben definierten Makros:

```
\def\SerieSep|{\Serie}
\obeyspaces\SerieSep|Umrandete Buchstaben und {W"orter}.\relax
```

Die zugehörige Ausgabe ist:



Was soll das `\obeyspaces` und warum habe ich noch so einen Wrapper `\SerieSep|` definiert? Im Wesentlichen ändert `\obeyspaces` den Catcode der Leerzeichen von 10 (=Blank Space) in 13 (=active), dadurch werden die Leerzeichen beim Einlesen der Argumente nicht mehr ignoriert und man erhält an deren Stelle die leeren Boxen. Trennt man jedoch das Kommando `\Serie` vom anschließenden `U` wie gewohnt durch ein Leerzeichen in der Form `\Serie U`, so erhält man unerwünschter Weise als erstes eine leere Box. Hier hilft das Kommando `\SerieSep`. Der Strich `|` von `\SerieSep|` wirkt als Separator zwischen dem Kommandonamen `SerieSep` und dem anschließenden `U`, da er Catcode `12` (=other) hat und nicht Catcode 11 (=letter). TEX parst beim Lesen von `\SerieSep|` den Strich `|` mit weg, er stört also nicht.

Noch ein ähnliches, sinnvolleres und sauberer programmiertes Makro:

```
\def\Serie#1#2{%
\bgroup%
\def\tmp{#2}%
\if\tmp\relax\egroup%
\def\next##1{}%
\else\egroup%
#1{#2}%
\let\next\Serie%
\fi%
\next#1}
```

Mit dieser `\Serie` kann man ein für ein Argument definiertes Makro, wie zum Beispiel `\fbox` nacheinander auf eine ganze Reihe von Argumenten anwenden. So liefert die Anweisung `\Serie\fbox Buchstaben{W"orter}\relax`

die Ausgabe B u c h s t a b e n Wörter. Die Gruppierung `\bgroup...\egroup` in der Definition von `\Serie` hält die temporär benutzte Variable `\tmp` lokal. Dabei zeigt sich noch eine interessante Wechselwirkung zwischen bedingten Anweisungen (`\if \else \fi`-Konstruktionen) und Gruppierungen. Abweichend von der Praxis in anderen Programmiersprachen haben bedingte Anweisgungen in TEX Vorrang gegenüber Gruppierungen. Das heißt, im `\if`-Zweig der bedingten Anweisgung wird das erste `\egroup` expandiert, während im `\else`-Zweig das zweite `\egroup` expandiert wird. Würde eine der zwei `\egroup`-Anweisungen fehlen, so würde TEX am Ende des Dokuments über nicht geschlossene Klammerebenen jammern. So kommt es, dass in diesem Makro zwei `\egroup`-Anweisungen auftauchen, obwohl es nur ein `\bgroup`-Kommando gibt.

Man kann auch nicht anstelle der zwei `\egroup`-Anweisungen eine hinter das `\fi` setzen, da dann `\def\next##1{}` und `\let\next\Serie` ungünstiger Weise ebenfalls lokal werden würden – unser rekursiver Algorithmus mit dem `\next` würde dann nicht mehr funktionieren. Außerdem würde dann `#1{#2}` ebenfalls lokal ausgeführt werden, was zum Beispiel bei der nächsten Anwendung (behebbare) Schwierigkeiten verursachen würde:

Die Zeilen

```
\def\Numbers#1{\expandafter\def\csname n#1\endcsname{\mathbb{#1}}}
\Serie\Numbers KNQRC\relax
$\nK\nN\nQ\nR\nC$
```

liefern die Ausgabe $\mathbb{KNQRC}$.

In der Definition von `\Numbers` wird durch `\expandafter` zuerst mit Hilfe von `\csname n#1\endcsname` ein Kommandosymbolname zusammengebastelt, ehe TEX `\def` ausführt. Der Kommandosymbolname wird aus dem Buchstaben `n` und dem Argument `#1` von `\Numbers` zusammengesetzt. So expandiert zum Beispiel `\Numbers{K}` zu `\def\nK{\mathbb{K}}`. Wäre `#1{#2}` in `\Serie` lokal, so gängen die ganzen schönen Definitionen aus `\Serie\Numbers KNQRC\relax` verloren, da sie alle lokal wären (das ließe sich brutal mit `\gdef` vermeiden, was in manchen Situationen jedoch auch nicht optimal ist).

5 Seitenüberschriften bei longtable

Torsten Wiebke warf die Frage auf, ob man innerhalb einer Longtable die Überschrift nach einem Seitenumbruch ändern kann. Dazu gibt es kein spezielles Makro in der Doku zu `longtable`. Inzwischen sind wir zu dem Schluss gekommen, dass es doch am einfachsten ist, die betreffende Tabelle in mehrere Longtables aufzuteilen. Das Problem ist also nicht mehr so dringend.

Jedenfalls haben wir zum Treffen eine Konstruktion der Form

```
\def\mytitle{Erster Titel}
\begin{longtable}
\mytitle
```

```
\endhead
Zeile 1\\
Zeile 2\\
u.s.w.
N"achster Tabellenteil
\def\mytitle{N"achster Tabellenteil}\\
Zeile 1\\
Zeile 2
\end{longtable}
```

ausprobiert. Trotzdem hat es als Seitenüberschriften der Tabelle immer „Erster Titel“ geliefert.

Durch einen Blick in `longtable.sty` wird auch relativ schnell klar, warum das nicht funktionieren kann. Mit dem Kommando `\endhead` wird nämlich der Eingabetext zwischen `\begin{longtable}` und `\endhead` in eine Box `\LT@head` gestopft, und danach wird zum Setzen der Seitenüberschriften nur noch die Box genutzt. Damit TEX die Boxdimensionen einfach ermitteln kann, expandiert es sofort alles vollständig, was in eine Box gestopft wird. Es steht also in der Box nur noch `Erster Titel` und nicht mehr `\mytitle` drin. Wir können also `\mytitle` ändern, wie wir lustig sind, es wird sich nie auf die Seitenüberschrift auswirken. Die einzige Chance, die Seitenüberschrift innerhalb der `\longtable` zu wechseln, ist, die Box neu zu setzen (`\setbox\LT@head\hbox{Neuer Titel}`). Aber das ist nicht so einfach, wie es zunächst aussieht, da dummerweise in die Box auch noch Formatierungsinformationen reingeschrieben werden, die schwer zu rekonsturieren sind.