DANTE

Deutschsprachige Anwendervereinigung TEX e.V.

Die TEXnische Komödie

Heft 2/1995 7. Jahrgang September 1995

Impressum

Beiträge sollten in Standard-LATEX-Quellcode an untenstehende Anschrift geschickt werden (entweder per e-mail oder auf Diskette). Sind spezielle Makros oder Schriften für die Bearbeitung erforderlich, so müssen auch diese mitgeliefert werden. Sind sie nicht bereits öffentlich zugänglich, sollten sie auf Anfrage Interessierten zugänglich gemacht werden.

Diese Ausgabe wurde mit Hilfe von `TeX, Version 3.1415 (C version 6.1)`, `LaTeX2e <1995/06/01> patch level 3`, `xdvik 18f` (für die Bildschirmdarstellung) und `dvipsk 5.58f` (für Korrektur und endgültige Belichtung) fertiggestellt.

Erscheinungsweise: vierteljährlich
Erscheinungsort: Heidelberg
Auflage: 3000

Herausgeber: DANTE, Deutschsprachige Anwendervereinigung TEX e.V.
Postfach 10 18 40
69008 Heidelberg
Tel.: 0 62 21/2 97 66
Fax: 0 62 21/16 79 06
e-mail: `dante@dante.de`

Druck: VOD Vereinigte Offsetdruckereien Mannheim Heidelberg
Handelsstr. 13
69214 Eppelheim
Tel.: 0 62 21/796-0
Fax: 0 62 21/796-137

Redaktionsschluß für Heft 3/1995: 15.10.1995

Editorial

Liebe Leserinnen und Leser,

es ist vollbracht! Oder doch zumindest in den Anfängen... Diese Ausgabe der Mitgliederzeitung ist mit neuen Makros erstellt, die (auch) mit LaTeX 2$_\varepsilon$ funktionieren. Großen Dank an den Autor Gerd Neugebauer, der die Umsetzung für uns erledigt hat. Jetzt muß ich mir hoffentlich keine Schmähbriefe mehr zu Gemüte führen, die (völlig zu Recht) anprangern, daß doch gerade diese Publikation mit der jeweils aktuellen Software erstellt werden sollte.

Aber es ist natürlich noch nicht alles erledigt. So sind die Makros zwar für den Hausgebrauch verwendbar, aber noch nicht so komplett durchgetestet, daß man sie guten Gewissens an die große TeX-Welt, sprich die Software-Server, übergeben könnte. Allerdings habe ich einen Großteil der Tests bereits mit der vorliegenden Ausgabe *nolens volens* durchgeführt. Auch will so etwas natürlich möglichst genau und verständlich dokumentiert sein. Wenn das aber erledigt ist, werden die Makros allen zugänglich gemacht. Das ist hauptsächlich für die Autoren gedacht, aber auch für diejenigen, die sich dafür interessieren, wie die Produktion der Mitgliederzeitung intern aussieht.

Der nächste Schritt ist dann die Überarbeitung des Layout. Das bedarf allerdings guter Überlegung und soll nicht überstürzt werden. Deshalb sollte niemand zu bald damit rechnen. Ansätze sind bereits vorhanden, Vorschläge werden jederzeit gern entgegengenommen. Auch was den Titel der Mitgliederzeitung angeht – soll der Titel geändert werden und wenn ja, wie könnte ein neuer lauten?

In der Hoffnung auf viele konstruktive Vorschläge verbleibe ich

Ihre Luzia Dietsche

Hinter der Bühne
Vereinsinternes

Grußwort

Liebe Mitglieder,

wie üblich möchte ich Sie über die wichtigsten Ereignisse in unserem Verein informieren. Dieses Mal wird mein Bericht allerdings sehr kurz ausfallen, da ich der Mitgliederversammlung in Berlin nicht vorgreifen will.

Bereits in der letzten Ausgabe der Mitgliederzeitung hatte ich das Vergnügen, ein neues Mitglied im technischen Beirat vorzustellen. Das Präsidium hat sich entschlossen, den Beirat um eine weitere Person zu erweitern. Das jüngste Mitglied in diesem Gremium ist Jürgen Unger. Vielen ist er schon bekannt, da er unsere Mailbox installiert und diese während der letzten Mitgliederversammlungen vorgestellt hat. Die Mailbox wird mittlerweile rege benutzt, dabei hat es sich herausgestellt, daß ein Verantwortlicher dafür von Nöten ist. Jürgen Unger war so leichtsinnig, auf die Frage, ob er zu einer solchen Aufgabe bereit sei, mit „ja“ zu antworten und ich freue mich, daß nun auch dieser Bereich abgedeckt ist.

Weniger erfreulich ist für viele das Thema der Mitgliedschaft bei der TEX Users Group (TUG). Einige (ehemalige) Mitglieder der TUG bekamen keine Rechnung für 1995 mit der bei späterer Nachfrage gegebenen Erklärung, daß dies wohl an der Post liegen müsse. Bei anderen war der Beitrag für 1994, den sie damals noch über unseren Verein bezahlen konnten, angeblich nie angekommen. In diesem Fall lautete die Auskunft der Verantwortlichen der TUG, daß DANTE e.V. die Information über den bezahlten Betrag wohl nicht an das Büro der TUG weitergegeben habe. Auf Grund der Unterlagen, die dem *Executive Director* übergeben worden waren und selbstverständlich DANTE e.V. noch in Kopie vorliegen, konnte Luzia Dietsche jedoch zweifelsfrei nachweisen, daß die entsprechenden Stellen bei der TUG die nötigen Daten erhalten haben müssen. Nach dieser Aufklärung stellte auch die TUG fest, daß ihr wohl ein Fehler unterlaufen war und korrigierte ihre vorherige Aussage gegenüber dem betroffenen Mitglied (ohne sich allerdings bei DANTE e.V. für die falschen Aussagen zu entschuldigen).

Luzia Dietsche hat mittlerweile ebenfalls Ihren Rücktritt aus dem *Board of Directors* (BoD) der TUG erklärt. Damit ist nun niemand mehr aus dem

Präsidium von DANTE e.V. im BoD vertreten. Ich danke Luzia für die Zeit, die sie im BoD verbracht und die Interessen der deutschsprachigen TUG-Mitglieder vertreten hat.

Glücklicherweise ist die 4allTEX-CD der *Nederlandstalige TEX Gebruikersgroep* (NTG) endlich wieder verfügbar, jetzt in der dritten Auflage, und wird von unseren Helfern bereits fleißig verschickt. Ich bitte alle, die auf diese CD warten, sich noch ein wenig zu gedulden, da wir eine sehr lange Warteliste haben, die nach und nach abgearbeitet werden muß.

Über alles weitere werde ich während der Mitgliederversammlung in Berlin berichten und hoffe, viele von Ihnen dort anzutreffen.

Bis zum nächsten Mal verbleibe ich damit, Ihr

Joachim Lammarsch

Von fremden Bühnen

Modifying LaTeX 2ε

LaTeX3 Project Team

Introduction

This document describes the principles underlying our policy on distribution and modification of the files comprising the LaTeX system. It has been produced as a result of wide-ranging discussions of the issues involved in the support and maintenance of a widely distributed document processing system used by diverse people for many applications. These discussions have involved users, maintainers of installations that support LaTeX and various types of organisations that distribute it. The discussions are continuing and we are hoping that this document will be widely distributed and make a useful contribution to the debate.

Our aim is to produce a system that users of all types can trust to fulfill their needs. Such a system must be stable and well-maintained. This implies that it must be reasonably easy to maintain it (otherwise it will simply not get maintained at all). So here is a summary of our basic philosophy:

> We believe that the freedom to rely on a widely-used standard for document interchange and formatting is as important as the freedom to experiment with the contents of files.
>
> We are therefore adopting a policy similar to that which Donald Knuth applies to modifications of the underlying TeX system: that certain files, together with their names, are part of the system and therefore cannot be changed unless the following conditions are met:
>
> - they are clearly marked as being no longer part of the standard system;
> - the name of the file is changed.

The system

In developing this philosophy, and the consequent limitations on how modifications of the system should be carried out, we were heavily influenced by the following facts concerning the current widespread and wide-ranging uses of the LaTeX system.

1. LaTeX is not just a document processing system; it also defines a language for document exchange.
2. The standard document class files, and some other files, also define a particular formatting of a document.
3. The packages that we maintain define a particular document interface and, in some cases, particular formatting of parts of a document.
4. The interfaces between different parts of the LaTeX system are very complex and it is therefore very difficult to check that a change to one file does not affect the functionality of both that file and also other parts of the system that are not obviously connected to the file that has been changed.

This leads us to the general principle that:

> with certain special exceptions, if you change the contents of a file then the changed version should have a different file name.

We certainly do not wish to prevent people from experimenting with the code in different ways and adapting it to their purposes. However, we are concerned that any distribution of modifications to the code should be very clearly identified as not being a part of the standard distribution. The exact wording and form of the distribution conditions is thus something that is flexible, but only within the constraint of keeping LaTeX as a standardised, reliable product for the purposes described above: the exchange and formatting of documents.

Some examples

Here we elaborate the arguments that have led us to the above conclusion.

Separate development considered harmful!

In many fields, the use of LaTeX as a language for communication is just as important as its capacity for fine typesetting; this is a very important consideration for a large population of authors, journal editors, archivists, etc.

Related to this issue of portability is the fact that the file names are part of the end-user syntax.

To take a real example, the LaTeX 'tools' collection contains the package 'array.sty'. A new user-level feature was added to this file at the end of 1994 and a document using this feature can contain the line:

```
\usepackage{array}[1994/10/16]
```

By supplying the optional argument, the document author is indicating that a version of the file `array.sty` dated no earlier than that date is required to run this document without error.

This feature would be totally worthless if we were to allow an alternative version of the array package to be distributed under the same name since it would mean that there would be in circulation files of a later date, but without the new feature. If the document were processed using this 'alternative array' then it would certainly produce 'undefined command' errors and would probably not be processable at all.

What's in a file-name?

In a pure markup language, such as SGML, it is reasonably clear that control over the final presentation lies with the receiver of a document and not with the author.

However, the way that LaTeX is often used in practice means that most people (at least when using the standard classes and packages) expect the formatting to be preserved when they send the document to another site.

For example, suppose, as is still the most common use of LaTeX in publishing, you produce a document for 'camera-ready-copy' using the class 'article' and that you carefully tune the formatting by, for example, adding some explicit line breaks etc, to ensure that it fits the 8 page limit set by the editor of a journal or proceedings.

It then gets sent to the editor or a referee who, without anyone knowing, has a non-standard version of the class file 'article' and so it then runs to 9 pages. The consequence of this will, at the least, be a lot of wasted time whilst everyone involved works out what has gone wrong; it will probably also lead to everyone blaming each other for something which was in fact caused by a misguided distribution policy.

It should also be noted that, for most people, the version of the class file 'article' that gets used is decided by a site maintainer or the compilers of a CD-ROM distribution. To most users, the symbols `article` in:

```
\documentclass{article}
```

are just as much part of LaTeX's syntax as are the symbols `12pt` in:

```
\hspace{12pt}
```

Thus they should both define a standard formatting rather than sometimes producing one more page or a 5pt larger space.

Users rely on the fact that the command (or menu item) 'LaTeX' produces a completely standard LaTeX, including the fact that 'article' is the 'standard article'. They would not be at all happy if the person who installed and maintains LaTeX for them were allowed to customise 'article' every second day so as (in her or his opinion) to improve the layout; or because another user wanted to write a document in a different language or typeset one with different fonts.

TeX itself

We have modelled our policies on those of the TeX system since this has for some time now been widely acknowledged as a very stable and high quality typesetting system.

The distribution policy set up by Donald Knuth for TeX has the following features:

- There is a clearly specified method for changing parts of the software by the use of 'change files'.
- Although arbitrary changes are allowed, the resulting program can be called TeX only if its functionality is precisely the same as that of TeX (i.e. neither less nor more) in all important areas.
- There are many files in the system that cannot be changed at all (without changing the name): examples are the file '`plain.tex`' and the files associated with fonts, including the Metafont source files.

Maintaining complexity

Our experience of maintaining LaTeX has shown us just how complex are the interactions between different parts of the system.

We have therefore, with lots of help from the bug reports you send in, developed a large suite of test files which we run to check the effects of every change we make. A non-negligible percentage of these test runs give unexpected results and hence show up some unexpected dependency in the system.

Some assurances

We are certainly not attempting to stop people reformatting LaTeX documents in any way they wish. There are many ways of customising incoming documents to your personal style that do not involve changing the contents of LaTeX's standard files; indeed, this freedom is one of the system's many advantages. The simplest way to achieve this is to replace

`\documentclass{article}` by `\documentclass{myart}`

Nor do we wish to discourage the production of new packages improving on the functionality or implementation of those we distribute. All we ask is that, in the best interests of all LaTeX users, you give your superbly improved class or package file some other name.

What do you think?

We are interested in your views on the issues raised in this document. The best way to let us know what you think, and to discuss your ideas with others, is to join the `LaTeX-L` mailing list and send your comments there. To subscribe to this list, mail to:

```
listserv@vm.urz.uni-heidelberg.de
```

the following one line message:

```
subscribe LATEX-L <your-first-name> <your-second-name>
```

Bretter, die die Welt bedeuten

Setzen russischer Textteile mit LaTeX

Gerd Neugebauer

In diesem Beitrag werden Möglichkeiten aufgezeigt, wie Texte mit kyrillischen Zeichen in einen normalen Text eingebracht werden können. Dabei werden insbesondere die kyrillischen Zeichensätze der University of Washington vorgestellt.

Einleitung

In der letzten Ausgabe wurde eine Möglichkeit aufgezeigt, wie kyrillische Buchstaben mit der `picture`-Umgebung gemalt werden können [4]. Diese Lösung hat den entscheidenden Nachteil, daß es damit mehr als mühsam ist, nicht nur einzelne Buchstaben, sondern Wörter oder ganze Sätze wiederzugeben.

Für diesen Zweck bietet es sich an, das Zeichnen mit METAFONT zu erledigen und das Zusammensetzen dem TeX-System zu überlassen. Bevor wir uns allerdings für einige Zeit verabschieden, um die METAFONT-Programme zu schreiben, haben wir nachgesehen, ob nicht irgend jemand vor uns bereits dieses Problem gelöst hat. Da es mit der CyrTUG eine russische TeX-Benutzergruppe gibt, sollte es auch eine annehmbare Lösung geben.

Im wesentlichen hat sich eine Lösung herauskristallisiert, die hier vorgestellt werden soll. Bevor wir uns jedoch in TeXnischen Tiefen verlieren, wollen wir noch etwas für unsere Bildung tun und schlagen im Duden [2] die relevanten Begriffe nach:

tran|skri|bie|ren ⟨lat.⟩ (*Sprachw.* einen Text in eine andere Schrift, z. B. eine phonet. Umschrift, übertragen; Wörter aus Sprachen, die keine Lateinschrift haben, annähernd lautgerecht in Lateinschrift wiedergeben [vgl. Transliteration]; *Musik* umsetzen);

Trans|li|te|ra|ti|on, die; -, -en ⟨lat.⟩ (*Sprachw.* buchstabengetreue Umsetzung eines Textes in eine andere Schrift [bes. aus nichtlateinischer in lat. Schrift] mit zusätzlichen Zeichen);

Das bedeutet, daß wir unter Umständen mehrere Möglichkeiten haben, wenn wir einen russischen Text mit lateinischen Buchstaben wiedergeben wollen. Um

Russisch	Mathematical Reviews	Zentralblatt f. Mathematik	Duden Transkription	Duden Transliteration
В в	v	v	w	v
Ё ё	ë	e	jo	e
Ж ж	zh	zh	sch (sh)	ž
Й й	ĭ	j	i	j
Х х	kh	kh	ch	h (ch)
Щ щ	shch	shch	schtsch	ŝ (šč)
Э э	è	eh	e	ė
Ю ю	yu	yu	ju	û (ju)
Я я	ya	ya	ja	â (ja)

Tabelle 1: Unterschiede von Transliterations- und Transkriptions-Systemen

diese Mehrdeutigkeiten zu vermeiden, muß man sich auf ein System einigen. Doch selbst das ist nicht so einfach. Nach einer kurzen Recherche habe ich schon vier Systeme vorliegen, die sich an einigen Stellen unterscheiden. Solche Unterschiede sind in der Tabelle 1 aufgezeigt. Die Systeme, die dort gezeigt werden, sind die folgenden: das System der *Mathematical Reviews*, des *Zentralblatt für Mathematik* und das des Duden. In diesem wird sowohl ein System zur Transkription als auch zur Transliteration vorgeschlagen.[1]

Wenn wir also einen Text haben, in dem solche transkribierten oder transliterierten russischen Textteile enthalten sind, dann kann es sinnvoll sein, die russischen Originaltexte in kyrillischer Schrift anzugeben, um eben solchen Unklarheiten vorzubeugen, die aus den Mehrdeutigkeiten der Darstellung in lateinischer Schrift entstehen. Genau dieses Problem werden wir nun in den folgenden Abschnitten angehen.

Die kyrillischen Fonts der University of Washington

In der Fontsammlung der American Mathematical Society ($\mathcal{AMS}$) ist eine Familie von kyrillischen Zeichensätzen enthalten. Dieses sind die `wncyr`-Fonts.

In Tabelle 2 sind einige Worte dargestellt. Diese sind in den Zeichensätzen `wncyr` (aufrecht), `wncyri` (kursiv), `wncb` (halbfett) und `wncss` (grotesk) gesetzt. Daneben ist die Transkription und die TeX-Eingabe zu sehen. Die Transkription

[1] Selbst dort sind noch mehrere Varianten erlaubt, die in der Tabelle in Klammern angegeben sind.

автор	*автор*	**автор**	автор	avtor	`avtor`	[Autor]
брат	*брат*	**брат**	брат	brat	`brat`	[Bruder]
вино	*вино*	**вино**	вино	vino	`vino`	[Wein]
гора	*гора*	**гора**	гора	gora	`gora`	[Berg]
док	*док*	**док**	док	dok	`dok`	[Dock]
еда	*еда*	**еда**	еда	eva	`eda`	[Essen]
жена	*жена*	**жена**	жена	zhena	`zhena`	[Frau]
земля	*земля*	**земля**	земля	zemlya	`zemlya`	[Erde]
идол	*идол*	**идол**	идол	idol	`idol`	[Götze]
мой	*мой*	**мой**	мой	moĭ	`mo\u\i`	[mein]
кадр	*кадр*	**кадр**	кадр	kadr	`kadr`	[Kader]
люди	*люди*	**люди**	люди	lyudi	`lyudi`	[Leute]
меню	*меню*	**меню**	меню	menyu	`menyu`	[Menü]
нет	*нет*	**нет**	нет	net	`net`	[nein]
око	*око*	**око**	око	oko	`oko`	[Auge]
поп	*поп*	**поп**	поп	pop	`pop`	[Pope]
работа	*работа*	**работа**	работа	rabota	`rabota`	[Arbeit]
сердце	*сердце*	**сердце**	сердце	serdtse	`serdtse`	[Herz]
театр	*театр*	**театр**	театр	teatr	`teatr`	[Theater]
уролог	*уролог*	**уролог**	уролог	urolog	`urolog`	[Urologe]
фабрика	*фабрика*	**фабрика**	фабрика	fabrika	`fabrika`	[Fabrik]
хлеб	*хлеб*	**хлеб**	хлеб	khleb	`khleb`	[Brot]
царь	*царь*	**царь**	царь	tsar′	`tsar\cprime`	[Zar]
чай	*чай*	**чай**	чай	chaĭ	`cha\u\i`	[Tee]
шнур	*шнур*	**шнур**	шнур	shnur	`shnur`	[Schnur]
щи	*щи*	**щи**	щи	shchi	`shchi`	[Kohlsuppe]
объект	*объект*	**объект**	объект	ob″ekt	`ob\cdprime ekt`	[Objekt]
мы	*мы*	**мы**	мы	my	`my`	[wir]
брать	*брать*	**брать**	брать	brat′	`brat\cprime`	[nehmen]
этюд	*этюд*	**этюд**	этюд	ètyud	`{\‘e}tyud`	[Etüde]
юг	*юг*	**юг**	юг	yug	`yug`	[Süden]
ягода	*ягода*	**ягода**	ягода	yagoda	`yagoda`	[Beere]

Tabelle 2: Transkription und TeX-Eingaben für `wncyr`

ist diejenige, wie sie von der $\mathcal{AMS}$ für die *Mathematical Reviews* verwendet wird. Eine vollständige Transkriptionstabelle ist in [1] enthalten.

Wie man leicht erkennen kann, sind die meisten Konstrukte direkte Übertragungen der Transkription. Da das Kyrillische mehr Zeichen kennt als das lateinische Alphabet, verbleiben einige Zeichen, die durch mehrere Buchstaben bzw. durch das Akzentuieren von einzelnen Buchstaben repräsentiert werden. Einige wenige Zeichen werden durch TeX-Befehle realisiert.

Uns interessiert nun, welchen Aufwand wir treiben müssen, um in den Genuß dieser Zeichensätze zu kommen. Erfreulicherweise hält er sich Aufwand in Grenzen.

Als erstes brauchen wir natürlich die Fonts selbst. Diese sind z. B. auf einem der CTAN-Server (*Comprehensive TeX Archive Network*) im Verzeichnis

```
/tex-archive/fonts/amsfonts/cyrillic
```

zu finden. Dort sind sowohl die METAFONT-Quellen als auch vorbereitete `tfm`- und `pk`-Dateien für verschiedene Ausgabegeräte vorhanden. Im einfachsten Fall kopieren wir diese Dateien einfach in ein Verzeichnis, in dem sie von den diversen Programmen gefunden werden können. Näheres dazu kann in dieser Allgemeinheit nicht gesagt werden, da die Details je nach Installation verschieden sind.

Als nächstes brauchen wir noch ein paar Makro-Definitionen, um die Zeichensätze verwenden zu können. Ein Teil der Arbeit steckt schon in der Datei `cyracc.def`, die im $\mathcal{AMS}$-Font-Paket enthalten ist, das man natürlich auch an eine geeignete Stelle kopieren muß.

Definitionen für plain-TeX und LaTeX 2.09

Beginnen wir mit dem (scheinbar) einfachen Fall. Leider stellt er sich dann doch nicht als ganz so einfach heraus, wir fangen jedoch trotzdem damit an. Er beschreibt eine Installation, die ohne *New Font Selection Scheme* (NFSS2) auskommen muß. Dabei müssen wir jeden Font einzeln behandeln. Dieser wird mit dem TeX-Kommando `\font` geladen.[2] Außerdem wäre es ganz praktisch, wenn man ein paar Font-Umschaltungsbefehle definieren würde, die man dann im laufenden Betrieb verwenden kann. Das Ganze kann dann etwa folgendermaßen aussehen:

```
\input{cyracc.def}
\font\tencyr=wncyr10
\def\cyr{\tencyr\cyracc}
\font\tencyri=wncyi10
\def\cyri{\tencyri\cyracc}
```

Damit stehen zwei Befehle zur Verfügung: `\cyr` und `\cyri`. `\cyr` schaltet auf den kyrillischen Zeichensatz der Größe 10 Punkt um. `\cyri` schaltet auf den entsprechenden kursiven (italic) Zeichensatz.

[2] Wir benutzen absichtlich nicht den LaTeX-Befehl `\newfont`, da wir auf diese Weise gleichzeitig palin-TeX und LaTeX 2.09 zusammen behandeln können.

Diese Definitionen sollten allerdings nur in einem lokalen Kontext angewendet werden, da sie einige Makros umdefinieren. Um also феникс zu erhalten, sollte man im Text `{\cyr feniks}` schreiben, nicht aber `\cyr feniks\rm`.

Der Nachteil dieser Methode ist sofort klar. Es muß jeder einzelne Font definiert werden. Diese Makros sind auch den „normalen“ Modifikatoren wie `\it` oder `\large` nicht zugänglich. Im nächsten Abschnitt werden noch einige Verbesserungen der dort eingeführten Makros vorgestellt, die analog auch für die oben genannten Definitionen vorgenommen werden können.

Definitionen für LaTeX 2ε

In LaTeX 2ε haben wir mit NFSS (Version 2) ein mächtiges Werkzeug zur Verfügung, das es uns erlaubt, neue Zeichensätze zu definieren, die auch Modifikatoren wie `\textit` oder `\large` zugänglich sind.

Dazu deklarieren wir zuerst eine neue Font-Familie und definieren die vorhanden Formen (shape) und Größen (size). Eine Font-Familie ist dabei eine Menge zusammengehöriger Fonts, wie z. B. `times` – oder eben `wncyr` in unserem Fall.

Die Formen entsprechen den Umschaltungen mittels `\textit`, `\textbf` und `\textsl`. Die Größen entsprechen den Befehlen `\tiny` bis `\Huge`.

```
\DeclareFontEncoding{OT2}{}{}
\DeclareFontFamily{OT2}{wncyr}{}
\DeclareFontShape{OT2}{wncyr}{m}{n}{
   <5> <6> <7> <8> <9> gen * wncyr
   <10> <10.95> <12> <14.4> <17.28> <20.74> <24.88> wncyr10 }{}
\DeclareFontShape{OT2}{wncyr}{bx}{n}{
   <5> <6> <7> <8> <9> gen * wncyb
   <10> <10.95> <12> <14.4> <17.28> <20.74> <24.88> wncyb10 }{}
\DeclareFontShape{OT2}{wncyr}{m}{it}{
   <5> <6> <7> <8> <9> gen * wncyi
   <10> <10.95> <12> <14.4> <17.28> <20.74> <24.88> wncyi10 }{}
\DeclareFontShape{OT2}{wncyr}{m}{sl}{ <-> sub * wncyr/m/it }{}
```

Die letzte Zeile stellt eine Substitution dar. Da zwar eine kursive, aber keine geneigte (slanted) Variante der `wncyr`-Fonts vorhanden ist, wird die geneigte durch die kursive Variante ersetzt.

Um andere Fälle zu behandeln, z. B. einen halbfetten Kursivzeichensatz, können diese Definitionen noch erweitert werden. Das bleibt allerdings dem Leser als Übung überlassen.

Als nächstes brauchen wir natürlich wieder die Makros aus den $\mathcal{AMS}$-Fonts. Also schreiben wir

```
\input{cyracc.def}
```

Schließlich brauchen wir noch ein Kommando, das auf den entsprechenden kyrillischen Zeichensatz umschaltet. Die einfachste Version davon ist die folgende

```
\newcommand{\cyr}{%
  \fontencoding{OT2}\fontfamily{wncyr}%
  \selectfont \cyracc}
```

Nun können wir im laufenden Text einfach `{\cyr Tsar\cprime}` schreiben und erhalten Царь. Wir sehen sofort, daß hier noch eine ziemlich umständliche Eingabe erforderlich ist. Die Transkription von ь ist einfach ′, wir müssen aber `\cprime` schreiben. Analog ist die Transkription von ъ einfach ″ (`\cdprime`). Also liegt es nahe, daß wir die Zeichen ' und " geeignet umdefinieren, um dann einfach `{\cyr Tsar'}` eingeben zu können. Das bedeutet insbesondere für das Zeichen " nur geringe Probleme, da die Bedeutung, die dieses Zeichen z. B. im German-Style hat, in diesen kyrillischen Passagen nicht benötigt wird.

Um den wenigen möglichen Problemen aus dem Weg zu gehen, fügen wir die folgenden Zeilen vor `\input{cyracc.def}` ein:

```
\chardef\TEMPCAT=\catcode`\"
\catcode`\"=12
```

Diese Zeilen bewirken, daß das " jetzt wirklich kein spezielles Zeichen mehr ist. Der alte Zustand von " wird in `\TEMPCAT` zwischengespeichert, um ihn am Ende wiederherstellen zu können.

Das weitere ist klar. Wir müssen die beiden Zeichen " und ' zu aktiven Zeichen machen. Danach können wir ein Makro an sie binden. Diese Makros sind natürlich `\cprime` bzw. `\cdprime`. Schließlich stellen wir den alten Zustand von " wieder her. Damit ergibt sich folgende Definition:

```
\begingroup
  \catcode`\'=\active \catcode`\"=\active
  \gdef\cyr{\fontencoding{OT2}\fontfamily{wncyr}%
    \selectfont \cyracc
    \catcode`\'=\active \let'=\cprime
    \catcode`\"=\active \let"=\cdprime
  }
\endgroup
\catcode`\"=\TEMPCAT
```

So weit, so gut. Wenn wir jetzt die vorangegangenen Definitionen in eine Datei schreiben, dann können wir die Erklärungen vergessen und uns an der Funktionalität erfreuen: wir schreiben `{\small\itshape\cyr tsar'}` und bekommen *царь*.

Alternativen

Natürlich ist es eine andere Art der Fragestellung, ob man nur ein paar Worte oder einen gesamten Text in kyrillischen Buchstaben setzen muß. Insbesondere muß hierbei natürlich das Trennproblem gelöst werden.

Dies geschieht mit Hilfe von russischen Trennmustertabellen. Diese Trennmuster verwenden Eingabekonventionen, die in einem erweiterten Font sowohl die lateinischen als auch die kyrillischen Zeichen enthalten. Das wird dadurch erreicht, daß sich in den ersten 128 Zeichen des Fonts die originalen ASCII- bzw. TeX-Zeichen befinden. In den oberen 128 Zeichen des Fonts werden die kyrillischen Zeichen untergebracht.

Das ist die Theorie. In der Praxis gibt es aber mehrere Kodierungen des oberen Zeichenbereiches. Zwei verbreitete Kodierungen sind KOI-8 und *Alternativnyj Variant* (AV). Den Trennmustern `rhyphen.tex` liegt die AV-Kodierung zugrunde. Mit diesen ist es möglich, lateinische und russische Texte zu mischen. Da dieses Verfahren die selben oberen Zeichenpositionen im Font wie z. B. die DC/EC- oder die ISO-Latin1-Fonts benutzt, ist klar, daß diese Zeichensätze nicht problemlos zusammen verwendet werden können. Die Eingabe wird dadurch erleichtert, daß bei geeigneten Tastatur-Treibern (für PC) die kyrillischen Buchstaben direkt eingegeben werden können.

Die `wncyr`-Fonts sind für das gerade skizzierte Vorgehen nicht gut geeignet, da in ihnen viele Ligaturen definiert sind, die zwar die Eingabe erleichtern, aber eine korrekte Trennung verhindern. Deshalb sind in dem $\mathcal{AMS}$-Fonts-Paket virtuelle Zeichensätze enthalten, die die Ligaturen nicht benutzen und verschiedene Kodierungen bereitstellen. In [3, Abschnitt 2.2.3] wird eine Erweiterung beschrieben, die beim Setzen von größeren Texten mit kyrillischen Buchstaben helfen soll.

Als Alternative zu den `wncyr`-Fonts gibt es noch die `cmcyr`-Fonts von Nana Glonti und Alexander Samarin. Diese sind auf einem der CTAN-Server in den Verzeichnissen

```
/tex-archive/fonts/cmcyr
/tex-archive/fonts/cmcyralt
```

АЖТОР	*АЖТОР*	*АЖТОР*	**АЖТОР**	АЖТОР	АЖТОР	[Autor]
БРАТ	*БРАТ*	*БРАТ*	**БРАТ**	БРАТ	БРАТ	[Bruder]
ЖИНО	*ЖИНО*	*ЖИНО*	**ЖИНО**	ЖИНО	ЖИНО	[Wein]
ГОРА	*ГОРА*	*ГОРА*	**ГОРА**	ГОРА	ГОРА	[Berg]
ДОК	*ДОК*	*ДОК*	**ДОК**	ДОК	ДОК	[Dock]
ЕДА	*ЕДА*	*ЕДА*	**ЕДА**	ЕДА	ЕДА	[Essen]
№ЕНА	*№ЕНА*	*№ЕНА*	**№ЕНА**	№ЕНА	№ЕНА	[Frau]
ЗЕМЛ	*ЗЕМЛ*	*ЗЕМЛ*	**ЗЕМЛ**	ЗЕМЛ	ЗЕМЛ	[Erde]
ИДОЛ	*ИДОЛ*	*ИДОЛ*	**ИДОЛ**	ИДОЛ	ИДОЛ	[Götze]
МО	*МО*	*МО*	**МО**	МО	МО	[mein]
КАДР	*КАДР*	*КАДР*	**КАДР**	КАДР	КАДР	[Kader]
Л ДИ	*Л ДИ*	*Л ДИ*	**Л ДИ**	Л ДИ	Л ДИ	[Leute]
МЕН	*МЕН*	*МЕН*	**МЕН**	МЕН	МЕН ü]	[Men
НЕТ	*НЕТ*	*НЕТ*	**НЕТ**	НЕТ	НЕТ	[nein]
ОКО	*ОКО*	*ОКО*	**ОКО**	ОКО	ОКО	[Auge]
ПОП	*ПОП*	*ПОП*	**ПОП**	ПОП	ПОП	[Pope]
РАБОТА	*РАБОТА*	*РАБОТА*	**РАБОТА**	РАБОТА	РАБОТА	[Arbeit]
СЕРДЦЕ	*СЕРДЦЕ*	*СЕРДЦЕ*	**СЕРДЦЕ**	СЕРДЦЕ	СЕРДЦЕ	[Herz]
ТЕАТР	*ТЕАТР*	*ТЕАТР*	**ТЕАТР**	ТЕАТР	ТЕАТР	[Theater]
УРОЛОГ	*УРОЛОГ*	*УРОЛОГ*	**УРОЛОГ**	УРОЛОГ	УРОЛОГ	[Urologe]
ФАБРИКА	*ФАБРИКА*	*ФАБРИКА*	**ФАБРИКА**	ФАБРИКА	ФАБРИКА	[Fabrik]
ХЛЕБ	*ХЛЕБ*	*ХЛЕБ*	**ХЛЕБ**	ХЛЕБ	ХЛЕБ	[Brot]
ЦАРЧ	*ЦАРЧ*	*ЦАРЧ*	**ЦАРЧ**	ЦАРЧ	ЦАРЧ	[Zar]
ЯА	*ЯА*	*ЯА*	**ЯА**	ЯА	ЯА	[Tee]
ЬНУР	*ЬНУР*	*ЬНУР*	**ЬНУР**	ЬНУР	ЬНУР	[Schnur]
ВИ	*ВИ*	*ВИ*	**ВИ**	ВИ	ВИ	[Kohlsuppe]
ОБЪЕКТ	*ОБЪЕКТ*	*ОБЪЕКТ*	**ОБЪЕКТ**	ОБЪЕКТ	ОБЪЕКТ	[Objekt]
МЫ	*МЫ*	*МЫ*	**МЫ**	МЫ	МЫ	[wir]
БРАТЧ	*БРАТЧ*	*БРАТЧ*	**БРАТЧ**	БРАТЧ	БРАТЧ	[nehmen]
Т Д	*Т Д*	*Т Д*	**Т Д**	Т Д	Т Дüde]	Т Д
Г	*Г*	*Г*	**Г**	Г	Г üden]	[S
ГОДА	*ГОДА*	*ГОДА*	**ГОДА**	ГОДА	ГОДА	[Beere]

Tabelle 3: Schriftprobe für `cmcyr`

zu finden. Hier sind allerdings nur die Fonts selbst und entsprechende virtuelle Fonts vorhanden. Die virtuellen Fonts sind von Alexander Harin und benutzen auch die AV-Kodierung. Die Eingabe über eine Transkription wird nicht unterstützt.

Wenn wir diese Font-Familie benutzen wollen, müssen wir also etwas Arbeit investieren, um die benötigten Makros bereitzustellen. Dafür bekommen wir aber eine vollständigere Palette von Zeichensätzen geliefert. Beispielsweise sind bei den `cmcyr`-Fonts sowohl geneigte als auch kursive Varianten dabei. Außerdem werden diese jeweils in normaler wie auch halbfetter Ausführung bereitgestellt.

Zum Vergleich ist in Tabelle 3 eine Schriftprobe mit der Fontfamilie `cmcyr` zu sehen. Dieselben Worte, die bereits in Tabelle 2 als Beispiel dienten, wurden auch hier benutzt. Sie sind jeweils in einer aufrechten, kursiven, geneigten, halbfetten, endstrichlosen und einer Schreibmaschinen-Variante zu sehen.

Schließlich gibt es auch noch die große Fülle diverser PostScript-Schriften. Darunter sind auch einige kyrillische Zeichensätze, wie `Compact`, `Pragmatica` oder `TimesET`. Das Erstellen der dafür benötigten virtuellen Fonts ist eine Aufgabe, die in einem eigenen Artikel beschrieben werden sollte. Mithin ist klar, daß diese Aufgabe nicht zwischen Tür und Angel zu erledigen ist. Vielleicht kann hierbei auch schon auf die Bemühungen anderer zurückgegriffen werden. Mir ist aber leider keine fertige Lösung bekannt.

Schlußbemerkung

Am Ende dieses Beitrages können wir bemerken, daß es recht einfach ist, kyrillische in lateinische Texte zu integrieren. Eine Lehre, die wir außerdem ziehen können, liegt darin, daß es sich immer lohnt, auf einem der CTAN-Server nachzusehen, bevor man sich an eine eigene Lösung macht. Aber eigentlich wußten wir das schon.

Danksagung

Dieser Beitrag verdankt seine Korrektheit zum Teil Burkhard Thiele. Vielen Dank für das geduldige Korrekturlesen und die Hinweise zum Russischen.

Literatur

[1] American Mathematical Society, *User's Guide to $\mathcal{AMS}$-Fonts Version 2.2*, 1995. Enthalten im $\mathcal{AMS}$-Fonts Paket.

[2] *Duden*, Band 1: Die deutsche Rechtschreibung, Dudenverlag, Mannheim, Leipzig, Wien, Zürich, 20. Auflage, 1991.

[3] Helmut Kopka, *LaTeX: Ergänzungen mit einer Einführung in METAFONT*, Addison-Wesley (Deutschland) GmbH, 1. Auflage, 1995.

[4] Siegfried Splett, Das russische Alphabet – mit „Bordmitteln“ erstellt, *Die TEXnische Komödie* 1/1995, 28–30.

Orale Spielereien mit TEX – Teil II

Bernd Raichle

„Zuerst laufen ihm die *Augen* über, dann bekommt er den *Mund* nicht voll genug, der *Magen* spielt dann nicht mehr mit. Das Ergebnis – er muß sich übergeben!“

Wenn man bei TEX-Stammtischen und -Tagungen diese Sprachfetzen hört, könnte man als Neuling zuerst meinen, daß da jemand gemeint ist, der sich bei einem der gemütlichen abendlichen Treffs etwas zuviel zugemutet hat. Dem ist jedoch nicht so: Der Satz beschreibt, wenn auch nicht ganz korrekt, das Ergebnis, den ein zerbrechlicher Befehl in LATEX auslösen kann, wenn er in einem „moving argument“ eines anderen Befehls nicht mit einem `\protect` geschützt wird.

Wie kommt es dazu, daß, wenn über TEX gesprochen wird, meist Analogien mit einem Lebewesen aufkommen und man TEXs Teile mit dem Verdauungstrakt vergleicht? Diesen Vergleich führt Donald E. Knuth im TEXbook [1, S. 38] selbst ein:

> [...] it is convenient to learn the concept by thinking of TEX as if it were a living organism. The individual lines of input in your files are seen only by TEX's “eyes” and “mouth”; but after that text has been gobbled up, it is sent to TEX's “stomach” [...]

Für diejenigen, die selbst nachlesen wollen: Diese Stellen sind im TEXbook über die Indexeinträge „anatomy of TEX“, „mouth“ und „stomach“ sehr einfach zu finden.

Das TEXbook beschreibt leider TEXs Arbeitsweise, insbesondere das Zusammenwirken der einzelnen Teile, nicht an einer einzigen Stelle. Stattdessen muß man sich das Gesamtbild in einzelnen Absätzen über mehrere Kapitel verteilt zusammensuchen. In der Serie „Orale Spielereien“ soll der Schwerpunkt auf dem „Mund“, TEXs Makroprozessor, liegen. Nachdem in der ersten Folge dem Mund schon die ersten kleineren Leckerbissen zum Naschen und Schlemmen angeboten wurden, will ich in der zweiten Folge daran gehen, TEX auseinanderzunehmen und nachzusehen, aus welchen Teilen TEX besteht und wie diese

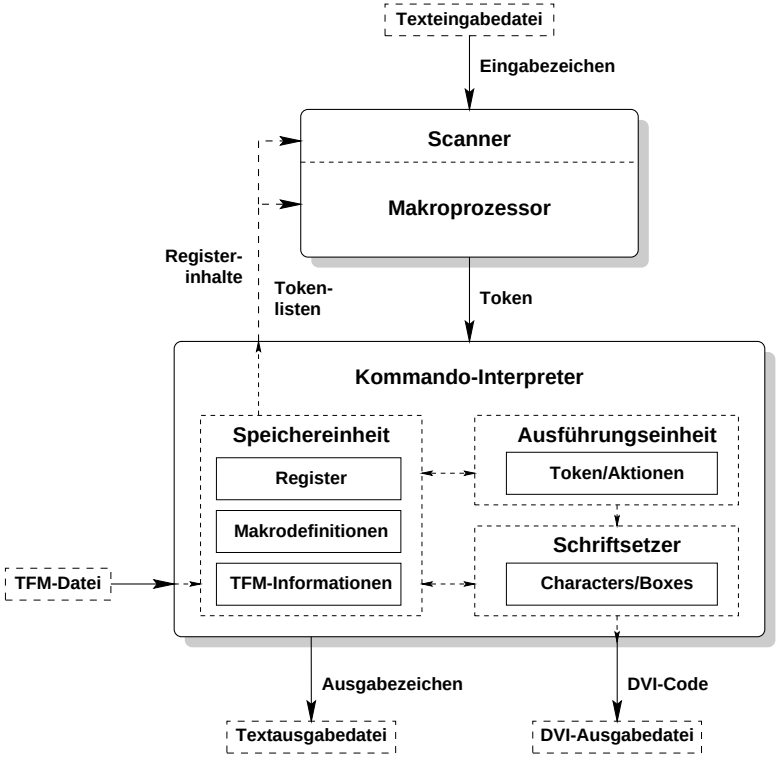


Abbildung 1: TEX und seine Bestandteile

bei der Verdauung der vom Benutzer vorgesetzten Dokumenthäppchen und mehrgängigen Menüs zusammenarbeiten.

Eine kleine Anatomiestunde – TEX auf dem Seziertisch

TEX kann man in folgende wichtige Bestandteile zerlegen [1, 2] (vgl. auch Abb. 1):

- Scanner: TEXs Augen (*eyes*) und Mund (*mouth*), die die Eingabe(-datei) Zeile für Zeile lesen und die Zeichen in Tokens umwandeln.
- Makroprozessor: Die Gurgel (*gullet*) ordnet durch Expansion Tokens in der durch den Mund erzeugten Tokenfolge um und kann dabei auch Tokens entfernen oder neue hinzufügen.

- Kommandointerpreter, zusammen mit Ausführungs- und Speichereinheit: TEXs zentrale Schaltstelle ist der Magen (*stomach*), in dem die ankommenden Tokens durch die Ausführungseinheit ausgewertet werden.
- Schriftsetzer: Im Darmtrakt (*bowels*) werden die Boxen zusammengebaut, mathematische Formeln aufgesammelt und anschließend gesetzt, Absätze umbrochen, alles zu Seiten zusammenmontiert und in die DVI-Ausgabedatei geschrieben.

Sehen wir uns diese Teile etwas genauer an:

Mund/„mouth“:

Beginnen wir mit dem Mund, da dieser eine Weichenfunktion für die Eingabe besitzt. Der Mund übergibt dem nachfolgenden Bearbeitungsschritt *genau ein einzelnes* Token. Dieses Token kann dabei aus unterschiedlichen Quellen stammen.

Durch eine vorangegangene Expansion eines Makros oder eines Tokenregisters kann eine Tokenliste auf dem Eingabestapel[3] liegen. Hier ist die Aufgabe einfach: Da das Token schon fertig geformt im Speicher vorliegt, kann dieses ohne weitere Umformung zurückgegeben werden.

Liegt keine Tokenliste auf dem Eingabestapel, so muß ein Token aus Zeichen der aktuell offenen Texteingabedatei gebildet werden. Dazu nimmt der Mund das nächste Zeichen aus dem Eingabepuffer. Jedem Eingabezeichen ist im Array `\catcode` ein *category code* zugeordnet, der die syntaktische Bedeutung des Zeichens für den Scan-Vorgang spezifiziert. Besitzt das Zeichen den *category code* „ignore“ oder „invalid“, so wird es ignoriert und evtl. eine Fehlermeldung ausgegeben. Ein Zeichen mit *category code* „blank space“ wird nur ignoriert, wenn zuvor schon ein Zeichen mit demselben *category code* gelesen wurde. Hat dieses Zeichen nun einen *category code* $\neq$ „escape“, so wird aus diesem Einzelzeichen ein Token gebildet. Hat es dagegen den *category code* „escape“ – in den meisten Formaten ist dies der Backslash \ – liest der Mund weitere Zeichen, aus denen er ein *control-sequence*-Token bildet.

Nun kann bei diesem Vorgang der Fall eintreten, daß der Eingabepuffer keine weiteren Zeichen enthält. In diesem Fall veranlaßt der Mund die Augen, die komplette nächste Zeile der Texteingabedatei zu lesen und im Eingabepuffer abzulegen.

[3] Auf den Eingabestapel und -puffer wird im Artikel „TEX capacity exceeded . . . “ in der nächsten Ausgabe näher eingegangen.

Augen/„eyes“:

Die Augen von TEX lesen eine komplette Zeile aus der Texteingabedatei und legen diese in den Eingabepuffer. Die Markierung für das Zeilenende hängt dabei vom Betriebssystem, bei Betriebssystemen, die unterschiedliche Dateiformate unterstützen, auch vom Dateityp ab. Daher ist dieser TEX-Teil implementierungsabhängig. Gleich beim Einlesen in diesen Puffer wandelt TEX die einzelnen Eingabezeichen von der vom Betriebssystem verwendeten Kodierung in die TEX-intern verwendete ASCII-Kodierung um. Dadurch kann man in den Makros Zeichencodes verwenden, die für jede Implementierung einheitlich sind. Abschließend entfernen TEXs Augen alle Leerzeichen am Ende einer Zeile und fügt dann an das Ende der Zeile ein Zeichen mit Code `\endlinechar` als TEX-intern verwendete Endekennung an.

Mund und Augen zusammen bilden den wichtigsten Teil des Eingabemechanismus, TEXs Scanner. Er kann durch die Register `\catcode`⟨*char_num*⟩ und `\endlinechar` an unterschiedliche Aufgaben angepaßt werden.

Gurgel/„gullet“:

Nachdem wir nun mit Augen und Mund dafür gesorgt haben, daß beständig Token für Token angefordert werden kann, kommen wir nun zum „Manipulator“ dieser Tokenfolge.

Die Gurgel, TEXs Makroprozessor, bekommt vom Mund nach und nach einzelne Tokens geliefert und formt diese Tokenfolge durch Expansion um. Jedem Token ist eine Bedeutung oder ein Wert zugeordnet, die in der *table of equivalents* der Speichereinheit nachgeschlagen werden. Ein Token kann expandierbar sein, d. h. ihm wurde vorher eine entsprechende Bedeutung zugeordnet. In diesem Fall wird das Token bei der Expansion im Makroprozessor durch eine dessen Bedeutung zugeordnete Tokenfolge ersetzt. Diese Tokenfolge ist der Expansionstext des Makros, die auf den zuvor erwähnten Eingabestapel des Mundes gelegt wird, so daß der Mund das nächste Token von dieser Tokenliste nehmen kann.

Die meisten expandierbaren Tokens, darunter alle Makros, besitzen keine oder bis zu neun Parameter. Besitzt es Parameter, werden vor dem Ablegen des Expansionstextes auf den Eingabestapel vom Mund weitere Tokens als Argumente gelesen und im Expansionstext an Stelle der Argumentplatzhalter `#1` ... `#9` eingesetzt.

TEXs Mund und Augen, die den Scanner bilden, und TEXs Gurgel, die den Makroprozessor realisiert, bieten dem Benutzer schon eine fertige Makrosprache

(Abb. 1, oberer Kasten). Mit dieser Makrosprache kann eine Textdatei gelesen werden, die durch den Scanner zu Tokens geformt wird. Die dadurch entstandene Tokenfolge wird durch Expansion, einer einfachen Text- bzw. genauer einer Tokenersetzung, umgeformt. Die umgeformte Tokenfolge wird danach ausgegeben, wobei die Ausgabe keine expandierbaren Tokens mehr enthält.

Der einzige Pferdefuß an dieser Makrosprache: Sie ist *noch* fest codiert und damit nicht an unterschiedliche Aufgaben anpaßbar. Man muß daher eine Möglichkeit schaffen, für den Scanner die syntaktische Bedeutung, den *category code* eines einzelnen Zeichens, und für den Makroprozessor die Bedeutung bzw. den Wert eines Tokens zu verändern. Beide Dinge werden in TEX durch den Magen realisiert (Abb. 1, unterer Kasten).

Magen/„stomach“:

Um ein Token als Eingabe zu lesen, verwendet TEXs Magen den zuvor beschriebenen Scanner und Makroprozessor. Da dadurch alle Tokens expandiert worden sind, tauchen als Eingabe im Magen nur Tokens auf, die Bedeutungen von nicht expandierbaren Primitiven besitzen.

Der Kommandointerpreter „Magen“ schlägt für jedes Token die zugeordnete Aktion nach und führt diese in der Ausführungseinheit aus. Diese Aktionen können zu Änderungen z. B. eines Registerwertes, der Bedeutung eines Token oder einer Makrodefinition in der Speichereinheit führen. Andere wichtige Aktionen führen zum Aufruf des Schriftsetzers, der z. B. ein weiteres Zeichen einem gerade im Aufbau befindlichen Absatz oder eine weitere Zeile der aktuellen Seite hinzufügt. Je nach Befehl müssen dabei noch weitere Tokens gelesen werden, die wiederum über Makroprozessor und Scanner angefordert und expandiert werden.

Die Speichereinheit ist ein wichtiger Teil TEXs, in der die Wertebelegungen der Register und die momentane Bedeutung einer *control sequence* oder eines *active character* abgelegt sind. Zusammenfassend greifen die verschiedenen TEX-Teile auf folgende Belegungen zu:

- Die Augen benötigen den Wert des speziellen Registers `\endlinechar`, um den Code des darin abgelegten Zeichens als TEX-interne Endekennung an das Ende einer Zeile anzufügen.
- Der Mund greift auf den Registersatz mit den *category codes* der einzelnen Zeichen zu, um mit dieser Information festzustellen, wie er aus einem oder mehreren Einzelzeichen ein Token zu formen hat.

- Die Gurgel, der Makroprozessor, benötigt die Bedeutungszuordnung eines Token und hier insbesondere die Expansionstexte und die Argumentzahl und -struktur der einzelnen Makros und *active characters*. Für die anderen expandierbaren Primitive ist meist der Zugriff auf den Wert eines Registers (z. B. `\the\count0`) notwendig.
- Der Magen bzw. die nachgeschalteten Teile greifen auf verschiedene Register, auf die Font-(TFM-)Informationen und auf die im Hauptspeicher (*main memory*) abgelegten Strukturen zu.

Darmtrakt/„bowels“:

Der „Schriftsetzer“ ist für ein Textformatierprogramm eine sehr wichtige Einheit. Die Ausführungseinheit ruft diesen Teil TEXs auf, um die vertikalen, horizontalen und mathematischen Listen und Boxen aufzubauen, Absätze umzubrechen und Tabellen zu formatieren. Aus diesen Teilen baut der Schriftsetzer komplette Seiten auf, die er Seite für Seite über den Rest des Darmtrakts als fertig gesetztes Dokument in die DVI-Ausgabedatei schreibt.

Nachdem wir TEX auseinandergenommen haben und die Einzelteile des gesamten Verdauungstraktes vor uns liegen, kommt einigen Leser bestimmt – so hoffe ich doch! – eine Frage in den Sinn. Wieso wird, wenn man vom Makroprozessor spricht, normalerweise die Gurgel verschwiegen und immer nur vom Mund gesprochen? Gerade wurde beschrieben, daß der Mund nur dafür zuständig ist, das nächste Token bereitzustellen. Die Gurgel darf dagegen den Hauptanteil beim ersten Schritt der Verdauung, die Expansion, leisten. Wie oft im Leben vereinfacht man die für das Verständnis notwendigen Dinge und faßt dazu mehrere Bestandteile zu größeren Einheiten zusammen. Bei der Makroexpansion arbeiten Mund und Gurgel sehr eng zusammen. Beide arbeiten zudem mit Tokens und Tokenlisten, so daß man die beiden Teile zusammen salopp als *Mund* bezeichnet. Genauso werden der Darmtrakt, der Kommandointerpreter, die Speicher- und die Ausführungseinheit als *Magen* zusammengefaßt, so daß man mit Mund und Magen bzw. mit Makroprozessor und Kommandointerpreter ein einfaches, aber für die meisten Zwecke ausreichendes Modell des Verdauungsapparates von TEX erhält.

Expandierbar oder nicht expandierbar? Das ist hier die Frage!

Zum weiteren Verständnis bleibt nun eine weitere wichtige Frage: Welche Tokens bzw. Primitive sind expandierbar und welche nicht? Welche Tokens werden damit vom Mund expandiert und welche gelangen zum Magen und werden dort interpretiert und ausgeführt?

Die Antwort eines TEX-Gurus, wenn man sich denn der Gnade einer Antwort würdig erweist, ist ein Verweis auf die Liste der expandierbaren Primitive im TEXbook [1, S. 212–215]. Nur die dort aufgeführten Primitive sind expandierbar und werden vom Mund verarbeitet, alle dort nicht aufgezählte Primitive gelangen in den Magen.

Diese Liste ist für die meisten Anfänger, mich eingeschlossen, nicht unbedingt die beste Referenz aller expandierbaren Primitive. Besonders zu Beginn schlägt man vorsichtshalber jedes Mal nach. Findet man das Token in dieser Liste nicht, ist man immer noch unsicher, ob es tatsächlich nicht expandierbar ist oder ob man es einfach nur übersehen hat.

Aus diesem Grunde habe ich mir folgende einfache Faustregel zurechtgelegt:

1. Alle Primitive und alle Makros werden als expandierbar angenommen.
2. Alle Primitive mit Seiteneffekten sind *nicht* expandierbar.
3. Die wichtigsten Ausnahmen von Punkt 2 bilden die Primitive `\relax`, `\lowercase`, `\uppercase` und `\ignorespaces`, die *nicht* expandierbar sind.

Im Gegensatz zu der Liste im TEXbook werden hier nicht die expandierbaren, sondern die nicht expandierbaren Primitive aufgezählt. Seiteneffekte sind im Sinne dieser Faustregel

- jedwede Art von Zuweisungen

  (z. B. `\count0=10`, `\chardef\ae="1A`, `\futurelet\nxt\do[`, `\aftergroup\egroup`, `\def\foo{\bar}`, `\let\foo=A`, ...),

- die Gruppierungsbefehle

  (z. B. `{`, `\begingroup`, `\bgroup`, ...),

- die Datei- und Textausgabeoperationen, die Zeichen in eine Textdatei schreiben oder aus einer Textdatei lesen

  (z. B. `\openin7=file.ext`, `\read7 to\foo`, `\message`, `\write`, `\show`, ...) und

- alle anderen Befehle, die eine Ausgabe in der DVI-Datei erzeugen oder diese verändern

  (z. B. `A`, `\hbox`, `\vrule`, `\par`, ...).

Natürlich gibt es neben den unter Punkt 3 genannten Primitiven weitere Ausnahmen, deren Klassifikation als ebensolche Primitive mit bzw. ohne Seiteneffekte schwieriger ist.[4] Darunter fallen u. a. die expandierbaren Primitive `\input` und `\endinput`, die dazu dienen, TEXs aktuelle Eingabedatei zu wechseln.

Bei der Entscheidung, ob nun ein Token expandierbar oder nicht expandierbar ist, muß man noch beachten, daß Tokens auch vom Mund bzw. Magen gelesen werden, um daraus das Argument eines Makros oder einer Aktion zu bilden. Beim Lesen eines Makroargumentes werden Tokens *nicht* expandiert. Ob ein Token beim Lesen der Argumente eines Primitivs expandiert wird, ist vom Primitiv selbst abhängig. Zum Beispiel werden die Tokens in den Argumenten von `\def` nicht expandiert, während beim `\edef` die Tokens, die den Expansionstext angeben, expandiert werden.

Eine gute Verdauung wünsch' ich!

Sehen wir uns jetzt einmal TEXs Arbeitsweise mit Hilfe eines einfachen Beispieles an. Unsere Eingabedatei auf einem PC enthält die folgende Zeile:

`\clubpenalty=1\number‘ä␣␣␣␣␣␣`$^{\texttt{C}}_{\texttt{R}}\,^{\texttt{N}}_{\texttt{L}}$

Den Verdauungsprozeß beginnt TEX gleich nach dem Start durch den Benutzer mit einem knurrenden Magen. Als erstes verlangt dieser von der Gurgel das nächste Token, damit er endlich was zum Verdauen bekommt. Die Gurgel meldet den Wunsch nach einem Token an den Mund. Da sowohl der Eingabestapel als auch der Eingabepuffer leer ist, werden die Augen aufgefordert, die nächste Zeile aus der momentanen Eingabedatei zu lesen.

Die Augen lesen nun die nächste Zeile bis zum Zeilenende, das mit den beiden beim verbreitetsten PC-Betriebssystem üblichen Zeichen *Carriage Return* $^{\texttt{C}}_{\texttt{R}}$ und *New Line* $^{\texttt{N}}_{\texttt{L}}$ markiert ist. Diese beiden Zeichen werden gelesen, jedoch *nicht* im Eingabepuffer abgelegt. Nach dem Lesen der Zeile werden dann alle Leerzeichen am Zeilenende entfernt und das Zeichen `\endlinechar`, normalerweise ein Ctrl-M (`^^M`), als letztes Zeichen an das Ende der Zeile in den Eingabepuffer eingefügt. Somit steht im Eingabepuffer folgende Zeile:

`\clubpenalty=1\number‘ä^^M`

Die Augen wandeln darüberhinaus die Codes der einzelnen Zeichen in der Zeile in ASCII um; nationale Sonderzeichen werden normalerweise in die Kodierung

[4] Die fortgeschritteneren Leser können bis zur nächsten Folge darüber rätseln, ob das Primitiv `\span` expandierbar oder nicht expandierbar ist. TEX-Anfänger werden dagegen erst einmal nachschlagen, ob es überhaupt ein solches Primitiv gibt und wenn ja, wofür es gedacht ist `:-)`.

ISO Latin-1 umgesetzt.[5] In unserem Beispiel wird der Umlaut `ä`, der in Codepage 850 den Zeichencode `"84` besitzt, durch das Zeichen mit dem TEX-internen Code `"E4` ersetzt.

Nachdem die Augen die nächste Zeile gelesen und mit den oben genannten Änderungen in den Eingabepuffer geschrieben haben, formt der Mund aus den ersten Zeichen ein Token. Da das erste Zeichen Backslash den *category code* „escape“ besitzt, werden weitere Zeichen gelesen und aus ihnen das *control-sequence*-Token `\clubpenalty` gebildet. Dieses Token ist nicht expandierbar, so daß die Gurgel es unverändert an den Magen weitergibt.

Das Primitiv `\clubpenalty` ist die linke Seite einer Zuweisung einer ganzen Zahl an das interne Register *clubpenalty* und benötigt somit die rechte Seite der Zuweisung als Argument. Vor den Tokens, die die ganze Zahl, das Argument der Zuweisung, bilden, darf in TEX optional ein Gleichheitszeichen stehen. Der Magen verlangt daher noch nach einer zu Beginn unbestimmten Anzahl weiterer Tokens, da erst beim Lesen des Arguments entschieden werden kann, wann es zu Ende ist. Dies ist für Anfänger verwirrend und selbst Fortgeschrittene stellt TEX hier oft eine Falle.[6]

Die nächsten beiden Token `=` und `1`, die der Mund formt, sind nicht expandierbar, so daß sie ohne Expansion die Gurgel passieren. Der Magen erkennt das Token `=` als das erwähnte optionale Gleichheitszeichen der Zuweisung; das Token `1` bildet die erste Ziffer der ganzen Zahl, deren Rest nun ziffernweise gelesen wird.

Das nächste vom Mund geformte Token `\number` ist expandierbar, so daß es diesmal nur bis zur Gurgel gelangt. Zur Expansion dieses expandierbaren Primitivs ist eine Zahl als Argument notwendig, für die der Mund noch weitere Tokens liefern muß. Das nachfolgende Token `‘` ist für TEX ein Kennzeichen, daß diese Zahl eine Konstante, genauer dem Zeichencode des nächsten Zeichens entsprechen soll.

In unserem Beispiel folgt das Token `ä` mit dem Code `"E4` bzw. `228`. Das expandierbare Primitiv `\number` erzeugt daraus die einzelnen drei Ziffertokens `2`, `2` und `8`, die als Tokenliste auf den Eingabestapel zurückgelegt werden. Da diese drei Tokens nicht expandierbar sind, erhält der Magen diese ohne weitere Expansion über den üblichen Weg durch den Mund und die Gurgel.

[5] Dies ist für alle Benutzer ratsam, die EC/DC-codierte Schriftfamilien ohne Kniffe und Probleme verwenden wollen. Einzige Ausnahme bildet das Zeichen `ß`, das statt des Codes `"FF` in ISO Latin-1 auf `"DF` umgesetzt sein sollte.

[6] Welches Ergebnis erzeugt die Eingabe „`\quad Pluspunkt:`“, wenn das Makro `\quad` durch „`\def\quad{\hskip1em}`“ definiert wurde?

Nachdem das letzte Ziffertoken `8` gelesen wurde, taucht das Token `^^M` in der Eingabe auf. Dieses Token ist nicht expandierbar und landet wiederum im Magen. Da es keine Ziffer oder ein anderer Bestandteil einer Zahl ist, beendet es die rechte Seite. Damit ist auch das benötigte Argument der Zuweisung gelesen, die jetzt ausgeführt werden kann: *clubpenalty* := 1228.

Dies soll als Lehrstoff für unsere erste Anatomiestunde ausreichen, oder?

* * *

Und nun zur Auflösung der in der letzten Folge gestellten Frage [4], welche Klammerpaare bei der Abarbeitung der Eingabe

```
\begingroup \uccode‘\m=‘\*
\def\x{\uppercase\bgroup\endgroup \def\asts}
\expandafter\x\expandafter{\romannumeral\the\n 000 }}
```

zusammengehören und in welchen Schritten das Makro `\asts` definiert wird.

Das essentielle Know-How dieses Makros steckt im Codestück

```
\uppercase\expandafter{\romannumeral\the\n 000 }
```

Der Rest darumherum dient nur dazu, die erzeugten „`m`“ durch Sternchen „`*`“ zu ersetzen. Doch jetzt zurück zur Erläuterung des komplizierteren Konstruktes.

In den ersten beiden Zeilen finden neben dem Öffnen einer neuen Gruppe zwei Zuweisungen statt. Zum einem wird das Sternchen „`*`“ als assoziierter Großbuchstabe des Zeichens „`m`“ deklariert, zum anderen wird das Makro `\x` definiert.

Erst in der dritten Zeile wird es interessant. Als erstes sollte auffallen, daß diese Zeile mehr schließende als öffnende Klammern enthält. Wie der Code trotzdem funktionieren kann, soll durch schrittweise Abarbeitung (für `\n` = 3) gezeigt werden:

```
1 \expandafter\x\expandafter{\romannumeral\the\n 000 }}
```

Durch die beiden `\expandafter` wird zuerst `\romannumeral` expandiert, bevor die Tokens `\x` und `{` betrachtet werden. `\romannumeral` benötigt als Argument eine Zahl, so daß `\the\n␣` zuerst zur Ziffer `3` expandiert wird. Diese Ziffer ergibt mit den drei nachfolgenden Ziffern `000` die von `\romannumeral` in römische Ziffern umzuwandelnde Zahl.

```
2 \x{\romannumeral 3000 }}
3 \x{mmm}}
```

Im nächsten Schritt wird das Makro `\x` expandiert.

```
4 \uppercase\bgroup\endgroup \def\asts{mmm}}
```

Jetzt wird ersichtlich, wo die fehlende öffnende Klammer steckt: Sie ist im Token `\bgroup` versteckt, dem die Bedeutung einer *impliziten* öffnenden Gruppenklammer zugeordnet ist. Da das Primitiv `\uppercase` für die öffnende Klammer seines Arguments auch die implizite Form erlaubt, kann man diese hier verwenden.

`\uppercase` liest eine durch Klammern begrenzte Tokenliste und ersetzt jedes Token, das einem normalen Zeichen entspricht, durch ein im Array `\uccode` angegebenes Zeichen. In unserem Fall sind nur die Tokens „`m`“ normale Zeichen. Durch die `\uccode`-Zuweisung für dieses Zeichen in der ersten Zeile unserer Eingabe ersetzt `\uppercase` diese durch das Sternchen „`*`“.

```
5 \endgroup \def\asts{***}
```

Im nächsten Schritt schließt `\endgroup` die Gruppe. Dadurch werden die vor der Gruppe gültigen Werte für `\uccode‘\m` und für die Bedeutung des Token `\x` wieder eingesetzt. Man hat durch diesen Kniff die beiden Änderungen lokal gehalten.

```
6 \def\asts{***}
```

Der letzte Schritt definiert dann das Makro `\asts` mit der geforderten Sternchenzahl.

Literatur

[1] Donald E. Knuth, *The TEXbook*, Addison-Wesley Publ., Reading, Mass., Juni 1991.

[2] Victor Eijkhout, The Structure of the TEX Processor, *TUGboat*, 12(1991), No. 2, S. 263–256. (Vorabdruck des ersten Kapitels aus [3].)

[3] Victor Eijkhout, *TEX by Topic*, Addison-Wesley Publ., Reading, Mass., 1991.

[4] Bernd Raichle, Orale Spielereien – Teil 1, *Die TEXnische Komödie*, 4/1994, S. 16–23.

Style-Files – leicht gemacht – zum Zweiten

Matthias Malek

Überlegungen

Mit Interesse las ich den Beitrag von Arne W. Steuer in [3] über die Entwicklung kleinerer Style-Files auch für Nicht-TEX-Cracks. Darum wurden die abgedruckten Makros auch gleich ausprobiert, wobei mir ein paar kleinere Macken aufgefallen sind.

- Der Vorschlag läßt nur maximal zwei Label auf einer DIN A4-Seite zu.
- Will man mehr als zwei Label auf eine Seite bringen, müssen im Style-File neue Makros für Titel, Untertitel, Inhalt und zum Erstellen des Label geschrieben werden.
- Die Definition der ganzen Seite als `picture`-Umgebung ist zu unflexibel, was die horizontale Positionierung der Label angeht.

Ausgehend von Herrn Steuers Artikel habe ich selbst ein wenig experimentiert und bin mit der Hilfe von [1, 2] auf folgende Lösung gestoßen.

Ausführung

Es sollen Label in den Maßen 60×200 mm und 35×200 mm auf DIN A4-Seiten ausgedruckt werden. Um mehrere Label auf einer Seite unterzubringen, ist es vorteilhaft, den gesamten Seitenbereich zur Verfügung zu haben. Basierend auf den Ausführungen in [2, Seite 285f]. benutzt der Autor einen hauseigenen `a4.sty`-File, der durch die Makros in Abbildung 1 nochmals für diesen Zweck modifiziert wird.

Um den unterschiedlichen Breiten der Etiketten Rechnung zu tragen, sollten verschiedene Schriftgrößen zur Verfügung gestellt werden (siehe Abbildung 2).

```
\setlength{\topmargin}{0pt}
\addtolength{\oddsidemargin}{-3cm}
\setlength{\textheight}{20cm} \setlength{\textwidth}{20cm}
\setlength{\headheight}{0pt}  \setlength{\headsep}{0pt}
\setlength{\footskip}{0pt}    \setlength{\footheight}{0pt}
\setlength{\unitlength}{1mm}  \pagestyle{empty}
```

Abbildung 1: Seitenanpassung

```
% Fonts für breite Etiketten
\newfont{\bigtitlefont}{cmss17 scaled \magstep5}
\newfont{\bigthemefont}{cmss17 scaled \magstep2}
\newfont{\bigcontentsfont}{cmss17 scaled \magstep1}
% Fonts für schmale Etiketten
\newfont{\smalltitlefont}{cmss12 scaled \magstep4}
\newfont{\smallthemefont}{cmss12 scaled \magstep3}
\newfont{\smallcontentsfont}{cmss12 scaled \magstephalf}
```

Abbildung 2: Fontdefinitionen

```
\renewcommand{\title}[1]{\renewcommand{\@title}{#1}}
\newcommand{\@title}{}
\newcommand{\theme}[1]{\renewcommand{\@theme}{#1}}
\newcommand{\@theme}{}
\newcommand{\contents}[1]{\renewcommand{\@contents}{#1}}
\newcommand{\@contents}{}
% Leeres Makro für Vergleiche:
\newcommand{\lng@empty}{}
```

Abbildung 3: Titel, Untertitel und Inhalt des Aktenordners als Makro

Der geneigte Leser kann hier natürlich eigene Vorstellungen verwirklichen. Zur Fontdefinition siehe auch [1, Seite 48 und Anhang C] und [2, Seite 218 ff].

Es findet keine Abfrage der Etikettenbreite mehr statt. Vielmehr werden durch die in Abbildung 6 gezeigten Makros die Befehle `\biglabel` zum Erzeugen eines 60 mm breiten und `\smalllabel` zum Erzeugen eines 35 mm schmalen Label bereitgestellt. Hier findet auch die Zuordnung der Schriftgröße statt, so daß sich die Definitionen aus [3, Seite 28, Abbildung 2] zu den in Abbildung 3 gezeigten vereinfacht.

Dadurch entfällt auch eine explizite Ablaufsteuerung (siehe [3, Seite 29, Abbildung 4]). Der Benutzer gibt nur den gewünschten Titel, Untertitel und Inhalt des Etiketts an. Die Kommandos `\biglabel` bzw. `\smalllabel` erzeugen dann die gewünschten Label. Nach jedem Labelbefehl können die Makros `\title`, `\theme` und `\contents` neu besetzt werden. Hierbei ist unbedingt darauf zu achten, daß zwischen den Befehlen keine Leerzeile oder ein `\par` bzw. `\\` steht, da sonst das folgende Etikett auf die nächste Seite ausgegeben wird. Dies hat seine Begründung darin, daß TEX die Boxen, in denen die Label erzeugt werden, als Zeile betrachtet. Eine Leerzeile oder ein `\par` stellt nun das Ende des laufenden Absatzes dar und TEX möchte das nächste Label in einem neuen Absatz

```
\title{Titel des Ordners}
\theme{Untertitel des Ordners}
\contents{Inhalt des Ordners}
%% Erzeuge mit den obigen Angaben ein 60mm breites Etikett:
\biglabel
%% Erzeuge mit den obigen Angaben ein 35mm breites Etikett:
\smalllabel
```

Abbildung 4: Befehlssatz des `labels.sty`

```
\documentstyle[a4,labels,german]{article}
\begin{document}
\title{Breiter Ordner}
\theme{Beispiele zu}
\contents{\item Style-Files \item Output-Routinen}
\biglabel
%% Hier keine Leerzeile einfügen !!
\title{Schmaler Ordner}
\theme{Style-Files}
\contents{\item labels.sty \item a4.sty}
\smalllabel
\end{document}
```

Abbildung 5: Beispiel zum `labels.sty`

formatieren. Da der Platz auf der aktuellen Seite hierfür jedoch nicht ausreichend ist, kommt es zu einem Seitenumbruch. Die in Abbildung 6 in den Zeilen 11 bis 14 definierten gerahmnten Boxen sind nach dem Gusto des Autors gestaltet und können selbstverständlich auch entfallen. Will man sich die Arbeit der Textpositionierung hiernach ersparen, sollte `\linethickness` in Zeile 10 auf 0 mm gesetzt werden. Die Rahmen sind dann unsichtbar. Die Abfrage in Zeile 20, ob ein Inhalt angegeben worden ist, verhindert lästige LaTeX-Warnungen wegen fehlender `\item`s. Das `\hskip 5mm` zum Schluß der Makros (Zeile 33) sorgt für einen einheitlichen Abstand zwischen den Label auf einer Seite.

Die wesentlichen Unterschiede zwischen `\biglabel` und `\smalllabel` sind die kleineren Fonts (Abbildung 6, Zeilen 3 bis 8), die Rahmenmaße der Boxen und die nach links verschobene `\parbox` (Abbildung 6, Zeilen 23 und 24), in die der Inhalt (`\contents`) ausgegeben wird. Die Linksverschiebung der `itemize`-Umgebung erschien dem Autor notwendig, um Platz für längere Inhaltsangaben zu schaffen. Da `itemize` per se rechts einrückt, schien es für das schmale Etikett etwas knapp zu sein.

```
\newcommand{\biglabel}{%
  %% 60mm width
 \def\@wtitle{\bigtitlefont
   \@title}%
 \def\@wtheme{\bigthemefont
   \@theme\\[20mm]}%
 \def\@wcontents{%
   \bigcontentsfont\@contents}%
 \begin{picture}(60,200)
  \linethickness{.3mm}
  \put(0,0){\framebox(60,200){}}
  \put(2,172){\framebox(56,26){%
    \@wtitle}}
  \put(2,2){\framebox(56,168){}}
  \put(2,155){%
   \begin{minipage}[t]{56mm}%
    \begin{center}
    \@wtheme
    \end{center}
    \ifx\lng@empty\@contents
     \relax
    \else
     %
     %
     \begin{itemize}
     \@wcontents
     \end{itemize}%
     %
    \fi
   \end{minipage}%
  }
 \end{picture}%
 \hskip 5mm\relax}
```

```
\newcommand{\smalllabel}{%
  %% 35mm width
 \def\@stitle{\smalltitlefont
   \@title}%
 \def\@stheme{\smallthemefont
   \@theme\\[20mm]}%
 \def\@scontents{%
   \smallcontentsfont\@contents}%
 \begin{picture}(35,200)
  \linethickness{.2mm}
  \put(0,0){\framebox(35,200){}}
  \put(1,177){\framebox(33,22){%
    \@stitle}}
  \put(1,1){\framebox(33,175){}}
  \put(1,155){%
   \begin{minipage}[t]{33mm}%
    \begin{center}
    \@stheme
    \end{center}
    \ifx\lng@empty\@contents
     \relax
    \else
     \hspace{-1.5em}%
     \parbox{33mm}{%
      \begin{itemize}
      \@scontents
      \end{itemize}%
     }%
    \fi
   \end{minipage}%
  }
 \end{picture}%
 \hskip 5mm\relax}
```

Abbildung 6: Erstellen eines breiten bzw. schmalen Etiketts mit Makros

Damit wären alle Makros für den Style-File `labels.sty` definiert. Wird er gemäß den LaTeX 2.09-Konventionen (siehe auch [1, Seite 19 ff]) als Option eingebunden, steht der Befehlssatz aus Abbildung 4 zur Verfügung. Ein kurzes Anwendungsbeispiel zeigt Abbildung 5.

Alle TeX-Wizards können jetzt unbesorgt die Augen wieder öffnen, ohne bleibende Schäden befürchten zu müssen. Denjenigen Lesern, die nun gerne tiefer

in die Materie der Layout-Entwicklung und der damit verbundenen Makrodefinitionen einsteigen möchten, sei die Lektüre von [2] empfohlen.

Literatur

[1] Helmut Kopka, *LaTeX – Eine Einführung*, Addison-Wesley Verlag, Bonn, 1988.

[2] Helmut Kopka, *LaTeX – Erweiterungsmöglichkeiten, mit einer Einführung in METAFONT*, Addison-Wesley Verlag, Bonn, 1990.

[3] Arne W. Steuer, *Style-Files – leicht gemacht*, *Die TeXnische Komödie* 3/1994, Seite 25–30.

Was Sie schon immer über TEX wissen wollten ...

Datumsangaben

Bernd Raichle

Anfang September kam über die Diskussionslisten die Anfrage

> Ich möchte gerne in meinem Dokument „Stand: September '95“ einfügen. Ähnlich dem `\today`-Kommando.

die sich hervorragend als Idee für den Tip verwenden läßt.

Für eine Lösung dieses Problems spickelt man einfach einmal in den Makrodateien nach, die am Datum „drehen“. So findet man im German-Style im Makro `\dategerman` folgende Redefinition des `\today`-Befehls:

```
\renewcommand{\today}{\number\day.~\ifcase\month \or
  Januar\or Februar\or M\"arz\or
  April\or Mai\or Juni\or
  Juli\or August\or September\or
  Oktober\or November\or Dezember\fi
  \space\number\year}
```

TEX legt in den vier Registern `\time`, `\day`, `\month` und `\year` das aktuelle Datum beim Start des Übersetzungslaufes ab. Diese Werte kann man durch das Primitiv `\number` entweder direkt ausgeben oder über If-Abfragen, arithmetische Operationen usw. weiter verarbeiten.

Als ersten Schritt zur Lösung kopieren wir dieses Makro, benennen es um und löschen die Tagesangabe:

```
\newcommand{\DatumMonatJahr}{\ifcase\month \or
  Januar\or Februar\or M\"arz\or
  April\or Mai\or Juni\or
  Juli\or August\or September\or
  Oktober\or November\or Dezember\fi
  ~\number\year}
```

Damit erhalten wir als Datumsausgabe statt „26. September 1995“ jetzt nur noch „September 1995“.

Im nächsten Schritt sollten die Hunderterstellen in der Jahresangabe entfallen. Die einfachste Lösung hierfür lautet:

```
\newcommand{\DatumMonatJahr}{\ifcase\month \or
  Januar\or Februar\or M\"arz\or
  April\or Mai\or Juni\or
  Juli\or August\or September\or
  Oktober\or November\or Dezember\fi
  ~'{\advance\year -1900\relax \number\year}}
```

Leider werden wir beim Wechsel vom Jahr 1999 auf das Jahr 2000 wie mit einem größeren Teil der momentan benutzten Software ein Problem bekommen. So ergibt sich für den 1. Januar 2001 mit unserer Definition die Ausgabe „Januar '101“. Daher muß eine bessere, wenn auch aufwendigere Lösung her:

```
\newcommand{\DatumMonatJahr}{\ifcase\month \or
  Januar\or Februar\or M\"arz\or
  April\or Mai\or Juni\or
  Juli\or August\or September\or
  Oktober\or November\or Dezember\fi
  ~'%
  \begingroup
    % Berechne Hunderterstellen...
    \count0=\year \divide\count0 100 \multiply\count0 100
    % ...und ziehe diese von der Jahreszahl ab.
    \count2=\year \advance\count2 -\count0
    % Gib dann die Jahreszahl mit zwei Stellen aus.
    \ifnum\count2<10 0\fi\number\count2
  \endgroup}
```

Die Berechnungen werden lokal in einer Gruppe mit den fast frei verfügbaren Registern `\count0` bis `\count9` durchgeführt. Man muß nur beachten, daß innerhalb der Gruppe kein Seitenumbruch auftreten sollte, da diese Register von TEX als Seitennummern in die DVI-Datei geschrieben werden. Außerdem sollten für lokale Berechnungen möglichst nur die geradzahligen Register 0, 2, 4, 6, 8 verwendet werden.

Damit haben wir nun mit „September '95“ unsere gewünschte Lösung und auch noch für den 1. Januar 2001 erhalten wir mit „Januar '01“ ein korrektes Ergebnis.

TEX-Beiprogramm

TEX unter OS/2 für akademische Minderheiten

Jürgen Hanneder

Als mir vor einigen Monaten die Festplatte mit zahlreichen Daten sowie dem gesamten TEX-System abstürzte und ich nur mit größter Mühe (wer macht schon Sicherungskopien von allen Konfigurationen?) zumindest die Daten wiederherstellen konnte, drängte es mich danach, der DOS-Welt den Rücken zu kehren. Mit dem Artikel über TEX und OS/2 im Hinterkopf wurde ich stolzer Besitzer des neuen Warp und verbrachte sodann einige Tage im Rechenzentrum, um die jeweils neueste Version von emTEX, OS/2-Emacs, AucTEX und diverser Hilfsprogramme von den bekannten Servern zu holen. Nach einigen Wochen der Einarbeitung sehe ich nun wieder Land und möchte potentiellen Umsteigern einige Hinweise zur Installation sowie Möglichkeiten zur Verarbeitung von exotischen Sonderzeichen geben.

Zunächst zur verwendeten Hardware: Ich besitze ein Notebook mit 4 MByte RAM und einer DX 486 CPU (40 MHz). Da ich auf die Lieferung der passenden Speichererweiterung nicht warten wollte, installierte ich das aufwendigere HPFS-Filesystems trotz Warnungen des Handbuchs über deutlich verminderte Leistung. Ich erwartete, daß das Arbeiten zur Qual werden würde, doch erstaunlicherweise ist es gerade noch erträglich. In dieser Konfiguration ist das Formatieren eines TEX-Dokumentes jedoch deutlich langsamer als unter DOS und die Bearbeitung von größeren Texten kann man nicht empfehlen. Dieser Hinweis betrifft natürlich nur die Ungeduldigen, die es mit der Minimal-Konfiguration des Arbeitsspeichers versuchen wollen – mit 8 MByte läßt es sich ganz anständig arbeiten. Die Festplatte sollte im übrigen deutlich über 100 MByte haben, denn TEX benötigt knapp 15 MByte, Emacs 30 MByte, die Zeichensätze (bei mir) knapp 10 MByte, das Verzeichnis OS/2 über 50 MByte, sodann noch Platz für die Auslagerungsdateien usw. Für eine gute TEX-Arbeitsumgebung benötigt man die folgende Ausstattung: Die aktuelle Version von emTEX[7] und die aktuelle Version von AucTEX[8].

[7] Zu finden bei `ftp.dante.de` im Verzeichnis `/tex-archive/systems/msdos/emtex`.

[8] Erhältlich bei `ftp.iesd.auc.dk` im Verzeichnis `/emacs-lisp/alpha/auctex.zip`. Zum Entpacken wird `unzip.exe` benötigt, das u. a. bei `ftp-os2.cdrom.com` als `/32bit/archiver/unz512x2.exe` zu finden ist.

Emacs benötigt noch einiges mehr. Wer den Internet-Zugang, wie ich, nicht am Arbeitsplatz hat, sollte sich zunächst den Hilfstext zur Installation von Emacs durchsehen, wo alle benötigten, aber nicht im Emacs-Paket enthaltenen Dateien aufgelistet sind. Nicht nötig ist meiner Erfahrung nach eine UNIX-kompatible Shell wie `bash` oder `bourne`. Mein AucTEX funktioniert einwandfrei mit `cmd.exe`.

Folgt man der emTEX- und Emacs-Installationsanleitung, die ebenfalls die Installation des AucTEX beschreibt, sollte man es ohne große Probleme zu einer lauffähigen Version bringen können. Die Batch-Datei zum Installieren von AucTEX muß in ein anderes Verzeichnis kopiert werden, um zu funktionieren, aber das sind „peanuts“ für den geplagten Installierer.

Bedeutend schwieriger war es, das ganze System schonend auf die Verarbeitung von Sonderzeichen vorzubereiten. Da ich bisher keinerlei Tips hierfür beschrieben gefunden habe, möchte ich hier kurz meine eigene – notgedrungen primitive – Lösung des Problems vorstellen.

TEX verarbeitet ja von Hause aus eine große Anzahl von Sonderzeichen und Akzente, wobei die `wsuipa`-Makros das Spektrum noch einmal stark erweitert haben. Doch jeder, der mehr als hundert mal `\=a` oder `\={\i}` getippt hat, bzw. versucht hat, einen Text, der zum Großteil aus solchen Steuerzeichen besteht, zu korrigieren, dürfte mit dieser Eingabemethode unzufrieden sein.

Die Lösung ist im Grunde recht einfach: Man erstellt eine Zeichenersetzungstabelle (`*.tcp` im emTEX, siehe Hilfstexte), die ein sonst nicht benötigtes ASCII-Zeichen beim Formatieren automatisch in eine Steuersequenz umwandelt. Danach weist man diesem ASCII-Zeichen ein Tastatur-Makro im Editor zu. Sodann verändert man den Bildschirmzeichensatz, so daß auf der ASCII-Position auch das benötigte Zeichen steht. Der Benutzer bekommt schließlich nichts mehr von der Verarbeitung der Steuerzeichen mit – sie verhalten sich in jeder Hinsicht wie „normale Zeichen“.[9]

Der erste Schritt stellt keinerlei Probleme dar, da die Hilfstexte ausführlich genug sind. Für die Anpassung von Emacs und des Bildschirmzeichensatzes unter OS/2 können jedoch einige Tips viel Zeit sparen.

Die Emacs-Installationsanleitung beschreibt die Einrichtung eines sonst nicht belegten Hyper-Keys (H). Man legt diesen „Key“ am besten auf die von Emacs

[9] Keine Lösung habe ich bisher für die Silbentrennung von Sprachen wie Sanskrit in lateinischer Umschrift gesehen; zwar gibt es eine „Silbentrennung“, die an jeder beliebigen Stelle trennt (`allhyph.tex`), doch da Worte mit Sonderzeichen ohnehin ausgenommen sind, macht dies wenig Sinn.

sonst nicht benötigte `AltGr`-Taste. Hierzu sucht man die Zeile in der Datei `site-start.el`, in der die `AltGr`-Taste dem `meta` zugeordnet wird und ersetzt `meta` durch `hyper`: `(altgr-modifier . hyper)`.

Es empfiehlt sich, eigene Makros auf eine Kombination mit dieser Taste zu legen, so daß die Standard-Tastaturbelegung des Emacs nicht verändert werden muß. Die notwendigen Schritte sind im Handbuch angegeben. Die meisten Kombinationen mit dieser Taste können nun mit eigenen Makros belegt werden. Da jedoch kein Handbuch angibt, wie man ein ASCII-Zeichen einem Tastatur-Makro zuweist, seien hier für alle, die (wie ich) des Emacs-Lisp unkundig sind, die Resultate eigener Experimente, aber vor allem Tips vom Autor der OS/2-Emacs-Version, zusammengefaßt.

Um ASCII 140 auf `AltGr-i` zu legen, fügt man folgende Zeile in die Startdatei ein:

```
(global-set-key [?\A-i] '(lambda () (interactive) (insert "\={\i}")))
```

Hier einige Beispiele für Abkürzungen, die für das Tastaturmakro verwendet werden können:[10]

```
[?\A-i]     AltGr-i
[?\A-I]     AltGr-Shift-i
"+"         +
[?\M-s]     Alt-s
"\C-f"      Alt-f
[C-f11]     Ctrl-F11
[M-f12]     Alt-F11
[C-f2 ?v]   Ctrl-F2, v
[C-f2 ?q]   Ctrl-F2, q
```

Weitere Beispiele finden sich in der Datei `emx-keys.el`, die die Standard-Tastaturbelegung definiert. Hieraus läßt sich auch ersehen, ob eigene Definitionen mit den bereits bestehenden kollidieren.

Der nächste Schritt, die Veränderung des Bildschirmzeichensatzes, ist weitaus schwieriger. Während es für DOS eine Unmenge an Editoren für Screenfonts gab, ist die Auswahl für OS/2 sehr klein: Es gibt nur einen und zwar im *Programmer's Toolkit for OS/2* von IBM. Dieser Zeichensatzeditor liest und erzeugt Dateien mit der Extension `fnt`. Um den Zeichensatz in OS/2 verwenden zu können, muß eine `fon`-Datei erzeugt werden, die dann, wie im Handbuch

[10] Einige Kombinationen funktionieren nicht, da sie vorbelegt sind, so z. B. `AltGr-q`.

beschrieben, in den Kreis der verfügbaren Zeichensätze aufgenommen werden kann.

Doch der Reihe nach. Zunächst kopiere man sich, wie in der Emacs-Installation angegeben, den kleinen Zeichensatz (`small.zip`). Er besteht aus drei Zeichensatzdateien für verschiedene Auflösungen und anderen Hilfsmitteln zum Erstellen eines OS/2-ladbaren Zeichensatzes. Zum Erstellen eines neuen Zeichensatzes benötigt man neben grafischem Geschick eine gewisse Begabung oder sehr viel Zeit im Umgang mit dem Fonteditor. Die Hilfstexte sind sicher nicht für Laien geschrieben. Man lädt also am besten eine bereits bestehende `fnt`-Datei aus dem `small.zip` Paket und nimmt diese als Grundlage. Für meinen Notebook-Bildschirm ist der 6-Punkt-Zeichensatz nur mit großer Anstrengung zu entziffern. Ich habe also alle Größenangaben im Header-Menu (außer denjenigen, die auf 0 eingestellt waren) mit einem feststehenden Faktor multipliziert und bei Fehlermeldungen kleinere Korrekturen angebracht. Resultat war ein größerer Zeichensatz mit einer notwendigerweise schlechten Auflösung.

Der nächste Schritt ist dann die Glättung des Zeichensatzes. Wenn alle Zeichen verändert und Akzente hinzugefügt sind, muß ein OS/2-ladbarer Zeichensatz erstellt werden. Sind alle Namen (`vga.fnt` etc.) beibehalten und der ganze *Programmer's Toolkit* vorschriftsmäßig installiert worden, kann die Batch-Datei `small.cmd` aufgerufen werden. Es wird nun eine neue `small.fon` Datei erstellt, die in OS/2 über das Font-Palette-Menu angemeldet werden muß.[11] Meldet man den Zeichensatz `small` dann in Emacs an (siehe Installationsanleitung)[12], kann man endlich anfangen zu arbeiten.

Kurioses aus dem Fundus

Gerd Neugebauer

Wenn man beginnt, die Ausstattung für ein Theaterstück zusammenzustellen, muß man aus Sparsamkeit wohl oder übel auf Teile aus dem Fundus zurückgreifen. Wenn man Glück hat, dann ist im Fundus das geeignete Requisit zu finden. Ist es dort nicht vorhanden, muß man sich überlegen, ob man es be-

[11] Falls man `small.fon` schon installiert hatte und ihn nach der Veränderung neu laden will, muß der alte `small.fon` aus der Liste der verfügbaren Zeichensätze gelöscht werden. Keinesfalls sollte er aus dem Verzeichnis manuell gelöscht werden. Bei einer Fehlermeldung, daß der Zeichensatz nicht löschbar ist, muß das OS/2 beendet und neu gestartet werden.

[12] Falls der Zeichensatz vergrößert wurde, muß die korrekte Größe, die im Menue „Header/Sizes“, Punkt „Average char width“ abzulesen ist, in der Datei `site-start.el` eingetragen werden.

schaffen oder herstellen kann. Manchmal findet man auch ein Teil, das auf den ersten Blick brauchbar erscheint. Erst nach vielen Aufführungen des Stückes sitzt ein Zuschauer im Publikum, dem auffällt, daß dieses Requisit nicht paßt.

Oh, Entschuldigung. Mir fällt gerade auf, daß ich nicht dabei bin, eine Theaterkritik zu schreiben, sondern etwas aus der TEX-Welt zu kritisieren habe. Also – das Stück, das hier aufgeführt wird, heißt LATEX und die Requisiten, um die es geht, sind einige der Font-Umschaltbefehle. Ich denke, daß alle, die LATEX verwenden, ziemlich schnell auf die folgenden Befehle stoßen:

`\scriptsize`, `\small`, `\normalsize`, `\large`

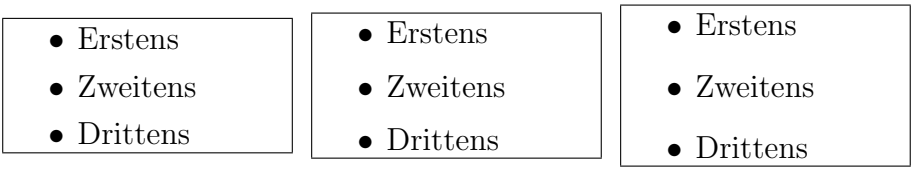
Viele glauben zu wissen, was diese Befehle zu bedeuten haben. Glaubt man [1, 2], dann dienen diese Befehle nur dazu, die Fontgröße umzuschalten. Das ist natürlich nicht genug. Zusätzlich muß auch noch der Abstand zwischen den Zeilen (`\baselineskip`) angepaßt werden, denn eine kleine Schrift erfordert einen geringeren Zeilenabstand als eine große Schrift. Das sollte es eigentlich gewesen sein. Insbesondere sollten alle Effekte, die z. B. durch `\small` entstehen, durch ein anschließendes `\large` auf die entsprechenden Werte gesetzt werden. Daß das nicht immer so ist, werden wir im folgenden sehen.

Wir wollen zeigen, daß die Font-Umschaltbefehle merkwürdige Nebeneffekte haben. Dazu benutzen wir immer das gleiche Schema

```
\begin{minipage}{.3\textwidth}
  \MACRO\large
  \begin{itemize}
    \item Erstens
    \item Zweitens
    \item Drittens
  \end{itemize}
\end{minipage}
```

Dabei verwenden wir anstelle von *\MACRO* jeweils `\footnotesize`, `\small` und `\normalsize`. Die Ergebnisse, die wir mit den Klassen[13] `book` oder `article` bei `11pt` erhalten, sehen wie unten dargestellt aus. Bei anderen Klassen oder Basisgrößen läßt sich der Effekt auch beobachten, jedoch zum Teil nicht so ausgeprägt.

[13] Das entsprechende gilt auch schon für die Stile in LATEX 2.09.



Um die Kuriosität offensichtlich werden zu lassen, haben wir die drei Experimente nebeneinander gesetzt und einzeln in `\fbox` eingeschlossen. Wir sehen also, daß die benutzten Befehle Seiteneffekte haben müssen, die nichts mit der puren Größenumschaltung zu tun haben. Ganz offensichtlich sind die Abstände zwischen den einzelnen Punkten verschieden, obwohl der Text in der gleichen Fontgröße gesetzt wurde.

Als (Theater-)Kritiker verfällt man allzu leicht in den Glauben, alles besser machen zu können als die Regisseure. Auch ich sehe einige Möglichkeiten, wie das Stück in Bezug auf den oben angeführten Kritikpunkt verbessert werden könnte und insbesondere, welche Details an den Requisiten mangelhaft sind. In weiser Zurückhaltung überlasse ich die Entscheidung jedoch anderen, welches die beste Methode ist, mit diesem Kritikpunkt umzugehen.

Vielleicht gehört dieses Requisit inzwischen zu dem Stück, da es doch bereits seit zwei Inszenierungen unter verschiedenen Regisseuren gleich geblieben ist? Vielleicht wird es in der nächsten Aufführung oder der nächsten Inszenierung verschwinden oder besser an das Stück angepaßt.

Zum Schluß noch ein versöhnliches Wort: Ich mag dieses Stück sehr und kann jedem nur empfehlen, dieses Theater zu besuchen – trotz der kleinen Mängel bei den Requisiten.

Literatur

[1] Leslie Lamport, *LaTeX: a document preparation system*, Addison-Wesley Publishing Company, Reading, Mass., 2. Aufl., 1994.

[2] Michel Goossens, Frank Mittelbach und Alexander Samarin, *The LaTeX companion*, Addison-Wesley Publishing Company, Reading, Mass., 1994.

The Underfull Badness Blues

by Frankeye Jones

(Sing to the tune of "Do Run Run")

I thought I put the backslashes in
I thought I did it right;
But when I tried to run the thing
The screen displayed this sight:
"Error, error, error," it said
So I've got no time to lose
My line's too long and my bracket's missing
I've got the underfull badness blues
(a-do-run-run-run, a do-run-run)

I can do most anything
With LaTeX as a tool
Make boxes, tables, Greek letters too
And I can alter the size of my pool
But wait! I got too excited again
It's the same old, not-good-news
My control sequences are in error again
I've got the underfull badness blues
(a-do-run-run-run, a do-run-run)

My life is like a LaTeX run
With trials and errors each day
The fates one minute are on my side
Then they slip and slide away
I lose my keys, I'm out of scope
I'm totally missing my cues
The cats have fleas and the water heater burst
I've got the overfull/underfull badness blues
(a-do-run-run-run, a do-run-run)

Rezensionen

Fachwörterbuch Kommunikationsdesign

Luzia Dietsche

Das Studium des Kommunikationsdesigns beschäftigt sich, laut Vorwort des Buches, damit, wie Transportmittel für Informationen gestaltet werden. Drucksachen soll eine gute Form gegeben werden, wobei die Form vom Inhalt abhängig ist. Auch Graphik-Design-Studierende kommen heutzutage nicht mehr ohne Computer aus und sind damit zwangsläufig mit einer (meist englischen) Fachsprache konfrontiert, die verstanden werden muß. Wie bei allen solchen Fachsprachen fehlt es jedoch an geeigneten Hilfsmitteln. Aus dieser Situation heraus entstand eine Diplomarbeit, aus der das hier besprochene Buch hervorgegangen ist.

Über 10 000 Begriffe sammelte die Autorin und bietet in dem zweisprachigen Buch Übersetzungen dafür an. Die Begriffe im deutsch-englischen Teil reichen von „abändern“ und „Abdecktusche“ über „Druckfehler“, „gestürzter Buchstabe“ und „Papierlaufrichtung“ bis hin zu „Zackenschnitt“ und „Zwischenzeile“. Im zweiten Teil kann man das Ganze dann selbstverständlich in der umgekehrten Richtung nachsehen. Es handelt sich nicht um ein Nachschlagewerk, das ausschließlich mit Textverarbeitung zu tun hat. Ganz im Gegenteil – auch Spezial- und Randgebiete des Kommunikationsdesigns sind berücksichtigt worden.

Besonderen Wert hat die Autorin auch auf die Form des Buches gelegt, eben auf das Design. Daraus ist ein ebenso funktionelles wie optisch ansprechendes Arbeitsmittel entstanden. Die beiden Teile sind durch unterschiedliche Farben voneinander abgehoben. Außerdem laufen beide Teile zur Mitte hin, das heißt, jeder Teil fängt an einem Deckblatt an. Das Buch wird komplett gedreht, um den jeweils anderen Teil zu verwenden. Es ist leicht erkennbar, ob der englische oder deutsche Teil aufgeschlagen wird, da das Deckblatt in der entsprechenden Sprache gestaltet ist. Zusammen mit Randmarken pro Buchstabe, klarer Seitenaufteilung und handlichem Format wird daraus ein gelungenes Ganzes.

Alles in allem handelt es sich um ein schön gestaltetes, nützliches Nachschlagewerk für alle, die auf internationaler Ebene im Bereich Typographie, Design und Grafik arbeiten.

Petra Wilhelm
Fachwörterbuch Kommunikationsdesign
Springer Verlag Berlin Heidelberg, 1995, (Edition Page)
ISBN 3-540-57779-3
48,– DM

Spielplan

Termine

14.–15.9.1995 13. Mitgliederversammlung DANTE e.V.
Humboldt-Universität, Berlin
Kontakt: Christiane Schöbel

ab 21.9.1995 Ausstellung: Illustrationen zu Dante Alighieris „Die göttliche Komödie“ von Salvador Dali
Katholisches Stadthaus Elberfeld
Laurentiusstraße 7
42103 Wuppertal

2.–5.10.1995 CyrTUG'95
Protvino (nahe Moskau), Rußland
Kontakt: Irina Makhovaya

27.–29.3.1996 DANTE'96 und 14. Mitgliederversammlung DANTE e.V.
Rechenzentrum der Universität Augsburg
Kontakt: Gerhard Wilhelms

7.–10.4.1996 EP 96 – International Conference on Electronic Documents, Document Manipulation and Document Dissemination
Xerox Research Center, Palo Alto, Kalifornien, USA
Kontakt: Xerox Corporation

Stammtische

In verschiedenen Städten im Einzugsbereich von DANTE e.V. finden regelmäßig Treffen von TEX-Anwendern statt, die für Jeden offen sind. Wer gerne auch einen solchen Termin anbieten möchte, um sich mit anderen TEXies auszutauschen, schickt einfach die Adresse der Ansprechperson, die Adresse des Treffpunktes und den Zeitpunkt des Treffens zur Veröffentlichung an die Redaktion.

12687 Berlin
Horst Szillat
Sella-Hasse-Str. 31
Tel.: 030/9 32 24 96 (Beantworter)
`szillat@berlin.snafu.de`
Gaststätte „Bärenschenke“
Friedrichstr. 124
Letzter Donnerstag im Monat, 19.00 Uhr

22527 Hamburg
Volker Huettenrauch
`volker_huettenrauch@hh.maus.de`
Hamb. Microcomputer-Hochschulgruppe
Grindelallee 143 (Hinterhof)
20146 Hamburg
Letzter Donnerstag im Monat, 18.00 Uhr

28759 Bremen
Martin Schröder
Tel.: 04 21/62 88 13
`l15d@alf.zfn.uni-bremen.de`
Universität Bremen, MZH 4. Stock
gegenüber den Fahrstühlen
Erster Donnerstag im Monat, 18.30 Uhr

35392 Gießen
Günter Partosch
HRZ der Justus-Liebig-Universität
Heinrich-Buff-Ring 44
`guenter.partosch@`
`hrz.uni-giessen.de`
„Licher Bierstuben“, Licher Straße
Letzter Montag im Monat, 19.30 Uhr

42283 Wuppertal
Andreas Schrell
Erlenstr. 1
Tel.: 02 02/50 23 54
`Andreas_Schrell@rs.maus.de`
Gasthaus „Yol“, Ernststr. 45
Zweiter Donnerstag im Monat, 19.30 Uhr

47226 Duisburg
Friedhelm Sowa
Rheinstr. 14
„Gatz an der Kö“, Königstraße 67
Dritter Dienstag im Monat, 19.30 Uhr

53111 Bonn
Uwe Münch
Schmittgasse 92
51143 Köln
Tel.: 0 22 03/8 20 62
`muench@ph-cip.uni-koeln.de`
„Anno“, Kölnstr. 47
Dritter Montag im Monat, 20.00 Uhr

69195 Wiesbaden
Christian Kayssner
Elsässer Platz 9
Tel.: 06 11/48 11 7
Andreas Klause, Elsässer Platz 3
Erster Montag im Monat, 20.00 Uhr

69008 Heidelberg
Luzia Dietsche
Tel.: 0 62 21/2 97 66
`dante@dante.de`
China-Restaurant Palast
Lessingstr. 36
Letzter Mittwoch im Monat, 20.00 Uhr

DANTE'96 in Augsburg

1. Einladung und Call for Papers

Wie schon mehrfach angekündigt, findet die 14. Mitgliederversammlung von DANTE e.V. und die TEX-Tagung DANTE'96

→ vom 27. bis 29. März 1996
an der Universität Augsburg

statt. Veranstalter sind gemeinsam das Rechenzentrum der Universität Augsburg und DANTE, Deutschsprachige Anwendervereinigung TEX e.V.

Am Mittwoch nachmittag sind Tutorien geplant. Die Mitgliederversammlung von DANTE e.V. wird voraussichtlich am Donnerstag vormittag stattfinden. Donnerstag nachmittag und Freitag sind für Vorträge, Diskussionen und Präsentationen vorgesehen.

- Alle, die einen *Vortrag* oder ein *Tutorium* halten oder eine *Diskussion* leiten wollen, werden gebeten, dies mit dem Formular „Anmeldung zur TEX-Tagung DANTE'96 in Augsburg"

  → möglichst bis 15. Dezember 1995

  anzumelden. Die Annahme von verspäteten Anmeldungen ist nur unter Vorbehalt möglich; diese werden im vorläufigen Tagungsprogramm möglicherweise nicht enthalten sein.

- Die Anmeldungen für die *Teilnahme* an der Tagung, die Reservierung der Hotelzimmer und die Bezahlung der Tagungsgebühr müssen

  → bis spätestens 1. März 1996

  erfolgen (Formulare werden separat verschickt). Die Tagungsgebühr wird 50,– DM für Mitglieder und 70,– DM für Nicht-Mitglieder betragen.

- Mit Fragen, Wünschen, Anregungen und wohlbegründeter Kritik wenden Sie sich bitte an

  Gerhard Wilhelms
  Lehrstuhl für Informatik I
  Universitätsstr. 14
  86159 Augsburg
  Tel.: 08 21/5 98 21 76
  Fax: 08 21/5 98 22 00
  email: `dante96@Uni-Augsburg.de`

oder an DANTE e.V. in Heidelberg.

- Alle Firmen und Institutionen, die ihre Produkte präsentieren bzw. die Tagung finanziell unterstützen wollen, werden gebeten, sich möglichst frühzeitig an dieselben Adressen zu wenden.

Wir hoffen, daß möglichst viele TEX-Interessierte unsere Veranstaltung in Augsburg besuchen werden und freuen uns auf einen erfolgreichen Tagungsverlauf.

Adressen

DANTE, Deutschsprachige Anwendervereinigung TEX e.V.
Postfach 10 18 40
69008 Heidelberg

Tel.: 0 62 21/2 97 66
Fax: 0 62 21/16 79 06
e-mail: `dante@dante.de`

Konten: Postgiroamt Karlsruhe
BLZ 660 100 75

2134 00-757	für Beiträge
2946 01-750	für Bücher und Disketten
1990 66-752	für Tagungen

Präsidium

Präsident:	Joachim Lammarsch	(`president@dante.de`)
Vizepräsident:	Uwe Untermarzoner	(`vice-president@dante.de`)
Schatzmeister:	Friedhelm Sowa	(`treasurer@dante.de`)
Schriftführerin:	Luzia Dietsche	(`secretary@dante.de`)

Server

ftp: `ftp.dante.de` [`129.206.100.192`]
e-mail: `ftpmail@dante.de`
gopher: `gopher.dante.de`
WWW: `http://www.dante.de/`
Mailbox: `0 62 21/16 84 26` (nur für Mitglieder)

Autoren/Organisatoren

Luzia Dietsche [3, 46]
siehe Seite 52

Jürgen Hanneder [38]
Burgstr. 38
35083 Mellnau

Rainer Hülse [45]
Wilhelmstr. 67b
47807 Krefeld
Tel.: 02151/305884

Joachim Lammarsch [4]
siehe Seite 54

LaTeX3 Project Team [6]
c/o Dr. Chris Rowley
The Open University
Parsifal College
Finchley Road
GB-London NW3 7BG
Tel.: +44/171/7940575
email: LTX3-Mgr@SHSU.edu

Irina Makhovaya [48]
CyrTUG, MIR Publishers
2 Pervyi Rizhskii Pereulok
Moskau, 129 820, Rußland
email: cyrtug@mir.msk.su

Matthias Malek [30]
Heckstr. 70
52080 Aachen

Gerd Neugebauer [11, 41]
Ödenburger Str. 16
64295 Darmstadt
email: gerd@imn.th-leipzig.de

Bernd Raichle [20, 36]
siehe Seite 54

Christiane Schöbel [48]
Humboldt-Universität
Unter den Linden 6
10099 Berlin
Tel.: 030/20 93 24 52
Fax: 030/20 93 29 59
email: dante-mv13@rz.hu-berlin.de

Gerhard Wilhelms [50]
Lehrstuhl für Informatik I
Universitätsstr. 14
86159 Augsburg
Tel.: 08 21/5 98 21 76
Fax: 08 21/5 98 22 00
email: dante96@Uni-Augsburg.de

Xerox Corporation [48]
XSoft Division
3400 Hillview Avenue
Palo Alto, California 94 304, USA
email: ep96@xsoft.xerox.com

Technischer Beirat

Zuschriften an die Koordinatoren werden in der Regel nur beantwortet, wenn ein ausreichend frankierter und adressierter Rückumschlag mitgeschickt wird. Die Koordinatoren sind nicht verpflichtet, auf jede Frage einzugehen.

Amiga
Markus Erlmeier
Postfach 415
84001 Landshut
Tel.: 0871/77939
Btx: 087177939-0001
MAUS: `Markus Erlmeier@LA`
FIDO: `2:2494/106.21`
Internet: `amiga@dante.de`

Atari
Stefan Lindner
Karolinenstr. 52b
90763 Fürth
`atari@dante.de`
oder
Lutz Birkhahn
Darfelder Str. 38
48727 Billerbeck
Tel.: 02543/4666
`atari@dante.de`

German-Style
Bernd Raichle
Stettener Str. 73
73732 Esslingen
`german@dante.de`

Graphik
Friedhelm Sowa
Heinr.-Heine Universität
Rechenzentrum
Universitätsstr. 1
40225 Düsseldorf
Tel.: 0211/3113913
`graphik@dante.de`

Lehrerfortbildung
Werner Burkhardt
Carl-Benz-Schule Mannheim
Neckarpromenade 23
68167 Mannheim
`lehrer@dante.de`

Macintosh
Lothar Meyer-Lerbs
Am Rüten 100
28357 Bremen
Tel.: 0421/252624
`macintosh@dante.de`

Mailbox von DANTE e.V.
Jürgen Unger
Ringstr. 24
64668 Rimbach
`mailbox@dante.de`

METAFONT
Jörg Knappen
Barbarossaring 43
55118 Mainz
`metafont@dante.de`

MVS
Joachim Lammarsch
Universitätsrechenzentrum
Im Neuenheimer Feld 293
69120 Heidelberg
`mvs@dante.de`
Vertreter:
Dr. Klaus Braune, s. UNIX

PubliCTEX
Dr. Peter Breitenlohner
Max-Planck-Institut für Physik
Postfach 40 12 12
80805 München
pc@dante.de

OS/2
Thomas Koch
Hauptstr. 367
53639 Königswinter
os2@dante.de

PostScript
Jürgen Glöckner
Ph.-Schmitt-Str. 8 b
69207 Sandhausen
Tel.: 06224/3750
postscript@dante.de

Server-Koordination
Dr. Rainer Schöpf
Zentrum für Datenverarbeitung
der Universität Mainz
Anselm-Franz-von-Bentzel-
Weg 12
55099 Mainz
server@dante.de

Treiberentwicklung und SGML
Joachim Schrod
Kranichweg 1
63322 Rödermark-Urberach
treiber@dante.de

UNIX
Dr. Klaus Braune
Universität Karlsruhe
Rechenzentrum
Zirkel 2
76128 Karlsruhe
Tel.: 0721/608-4031
unix@dante.de

VAX/VMS
Gerhard Friesland-Köpke
Universität Hamburg
FB Informatik
Vogt-Kölln-Str. 30
22527 Hamburg
vms@dante.de

Verlag und Buchhandel
Christa Loeser
Intern. Thomson Publ. GmbH
Trübnerstr. 38
69121 Heidelberg
Tel.: 06221/400177
Fax: 06221/472909
verlage@dante.de

VM
Dr. Georg Bayer
TU Braunschweig
Rechenzentrum
Postfach 3329
38023 Braunschweig
vm@dante.de

Inhalt Heft 2/1995