Die TEXnische Komödie

dante
Deutschsprachige
Anwendervereinigung TEX e.V.

28. Jahrgang Heft 3/2016 August 2016

3/2016

Impressum

Beiträge sollten in Standard-LATEX-Quellcode unter Verwendung der Dokumentenklasse `dtk` erstellt und per E-Mail oder Datenträger (CD/DVD) an untenstehende Adresse der Redaktion geschickt werden. Sind spezielle Makros, LATEX-Pakete oder Schriften notwendig, so müssen auch diese komplett mitgeliefert werden. Außerdem müssen sie auf Anfrage Interessierten zugänglich gemacht werden. Weitere Informationen für Autoren findet man auf der Projektseite `http://projekte.dante.de/DTK/AutorInfo` von DANTE e.V.

Diese Ausgabe wurde mit `LuaTeX, Version 0.95.0 (TeX Live 2016)` erstellt. Als Standardschriften kamen Libertinus Serif, Libertinus Sans, Anonymous Pro und Libertinus Math zum Einsatz.

Erscheinungsweise: vierteljährlich
Erscheinungsort: Heidelberg
Auflage: 2400
Herausgeber: DANTE, Deutschsprachige Anwendervereinigung TEX e.V.
Postfach 10 18 40
69008 Heidelberg

E-Mail: `dante@dante.de` (DANTE e.V.)
`dtkred@dante.de` (Redaktion)

Druck: Konrad Triltsch Print und digitale Medien GmbH
Johannes-Gutenberg-Str. 1–3, 97199 Ochsenfurt-Hohestadt

Redaktion: Herbert Voß (verantwortlicher Redakteur)
Mitarbeit: Gert Ingold Eberhard Lisse Rolf Niepraschk
Christine Römer

Redaktionsschluss für Heft 4/2016: 15. Oktober 2016 ISSN 1434-5897

Editorial

Liebe Leserinnen und liebe Leser,

diese Ausgabe unserer Zeitschrift erreicht Sie wieder im gewohnten Rythmus. Wir wollen versuchen, zukünftig die DVD TEX-Collection einen Monat früher oder später zu produzieren. Dann passt der Versand besser in unseren zeitlichen Ablauf.

Diese Ausgabe widmet sich praktisch der Skriptsprache Python. Damit ist nicht die Anwendung des Paketes PythonTEX (`http://ctan.org/pkg/pythontex`) gemeint, sondern die sogenannte Vor- oder Nachbearbeitung von TEX-Dateien. Die Nachbearbeitung geschieht auch bei der Erstellung einer Bibliografie oder eines Indexes und ist ein großer Vorteil; es lassen sich beliebige Modifikationen oder Erweiterungen am vorhandenen Quellcode oder den erzeugten externen Dateien vornehmen. Seltener wird dagegen ein Quellcode vorab bearbeitet, was aber ebenso möglich ist. Diese Bearbeitungen kann man mit verschiedenen Skript- oder Programmiersprachen vornehmen. Hier werden zwei Beispiele mit Python gezeigt.

Vor längerer Zeit fand ich einen interessanten Beitrag in der Newsgruppe `de.comp.text.tex` zur Funktionsweise von TEX. Ich hatte ihn zwischenzeitlich aus den Augen verloren, sodass er erst jetzt in der Rubrik »Im Netz gefunden« erscheint.

Ich wünsche Ihnen wie immer viel Spaß beim Lesen und verbleibe

mit TEXnischen Grüßen,

Ihr Herbert Voß

Hinter der Bühne

Vereinsinternes

Grußwort

Liebe Mitglieder,

in diesem Jahr scheint das Sommerloch wesentlich kleiner auszufallen als in den vergangenen Jahren. Vermutlich liegt dies an den sportlichen Großereignissen, nicht zuletzt aber wohl auch an der doch etwas angespannten politischen (Gemenge-) Lage. Ich hoffe dennoch, dass Sie Ihren Urlaub genießen konnten oder wie ich noch vor sich haben.

Wir nähern uns mit großen Schritten der Herbsttagung in Göttingen. Nachdem der Zuspruch im letzten Jahr vermeintlich wegen der etwas größeren Entfernung (Veranstaltungsort war Graz) ein wenig zu wünschen übrig ließ, zeichnet sich auch in diesem Jahr – zumindest was die Vortragsbeteiligung angeht – ein ähnliches Bild ab.

Mir gibt diese Entwicklung zu denken. Auch im Frühjahr müssen wir um jeden Vortrag »kämpfen«, um den gewohnten Programmumfang zu halten. Aber vielleicht ist das ja genau das Problem? Ist der Punkt erreicht, um über unsere Veranstaltungsangebote und den Umfang nicht nur kritisch nachzudenken, sondern auch Konsequenzen zu ziehen?

Ich persönlich fände es schade, wenn wir nur noch einmal im Jahr eine Tagung anbieten würden. Andererseits ist eine »erweiterte Vorstandssitzung« im Herbst auch nicht im Sinne des Erfinders.

Bei 2000 Mitgliedern und vielleicht 100 bis 120 Personen, die in den letzten Jahren die Tagungen besucht haben, ist es natürlich schon ein spannende Frage, wie wir die anderen knapp 1900 Mitgliedern vom Besuch einer Tagung überzeugen könnten. Schreiben Sie mir doch einfach, was Sie abhält oder was Ihnen fehlt.

Natürlich können Sie sich auch noch kurzfristig zu einer Wochenendreise nach Göttingen entschließen. Wir freuen uns über alle Mitglieder und Gäste unserer Tagung.

Wenn es um alternative Tagungskonzepte geht, ist die Auswahl heutzutage sehr groß. Gerne erinnere ich in diesem Zusammenhang an die Idee von Doris Behrendt,

im Rahmen des CCC-Camps eine »TEX-Stadt« zu errichten (vgl. mein Grußwort in der DTK 15-4).

Neben finanziellem Engagement des Vereins ist vor allem persönlicher Einsatz vor Ort gefragt. Daher lade ich Sie schon heute herzlich ein, sich bei grundsätzlichem Interesse an mich oder direkt an Doris (doris@dante.de) zu wenden. Nur wenn es ausreichend Personal gibt, lässt sich ein solch anspruchsvolles Event überhaupt sinnvoll planen und gestalten.

Apropos Mitmachen: Wir suchen auch für das kommende Jahr wieder einen Ehrenpreisträger (oder eine Gruppe), der sich durch regelmäßiges und überdurchschnittliches Engagement in der TEX-Community hervorgetan hat. Der Vorstand möchte gerne Vorschläge aus der Mitgliederschaft aufgreifen, daher hoffe ich, dass mein vorgezogenen Aufruf vielleicht schon erste Früchte trägt.

Wir sind als TEX-Nutzer in der glücklichen Lage, viele hoch motivierte Freiwillige in unseren Reihen zu haben, die seit drei Jahrzehnten für stetige Verbesserungen und Erweiterungen sorgen. Der mit 500 Euro dotierte Ehrenpreis ist finanziell gesehen sicherlich eine zu vernachlässigende Kompensation – es kommt der überwiegenden Mehrheit der Beteiligten ja auch nicht auf Ihren Verdienst an, sondern den Beitrag, den sie zum Gelingen von TEX beitragen können und die damit verbundene immaterielle Anerkennung. Letztere war die initiale Motivation für den Ehrenpreis.

Warum führe ich dies an dieser Stelle (nochmals) aus? DANTE e.V. fördert Projekte und hat sich dabei schon vor knapp 15 Jahren gegen die Übernahme von Personalkosten entschieden. Vielmehr sollen Sach- und Reisekosten im Mittelpunkt einer Förderung stehen.

Das Ganze kann man als Resultat schlechter Erfahrungen in der Vergangenheit sehen, man kann es aber auch als Anerkennung der Freiwilligenkultur im TEX-Bereich begreifen, die es zu bewahren gilt.

In den vergangenen Jahren gab es überwiegend Projektanträge, deren Ziele bzw. Ergebnisse unbestritten nützlich sind, deren Förderung aber nicht immer eindeutig und ausschließlich auf Sachmittel zielte.

Der Vorstand hat daher in der abgelaufenen Wahlperiode die Konzentration auf Sachmittel bekräftigt und beschlossen, bei künftigen Projektanträgen höhere Hürden aufzustellen.

Auch wenn dies nicht auf die ungeteilte Zustimmung aller Beteiligten stößt, bin ich überzeugt, dass es der richtige Weg ist.

In diesem Sinne wünsche ich Ihnen viel Vergnügen bei der weiteren Lektüre.

Herzlichst Ihr/Euer
Martin Sievers

DANTE e.V. sucht Veranstalter für Tagungen

Wir suchen für kommende Tagungen 2018 (gerne auch schon für 2019) noch Ausrichter, die DANTE e.V. Räumlichkeiten zur Verfügung stellen und die Organisation vor Ort übernehmen.

Wer kann sich melden? Angebote aus dem gesamten deutschsprachigen Raum sind grundsätzlich willkommen. Gerne sind wir an neuen Orten zu Gast. »Traditionsorte« sind aber auch immer eine Reise wert.

Was muss ich zur Verfügung stellen? Wir benötigen für Herbsttagungen einen Vortragsraum für etwa 40 Personen, für Frühjahrstagungen sollte der Raum ein wenig größer sein. Zudem wird für Frühjahrstagungen ein zweiter Raum als Tagungsbüro benötigt.

Wieviele Personen sind vor Ort nötig? Man kann eine Tagung sicherlich auch alleine organisieren. Es hat sich aber gezeigt, dass die Unterstützung durch ein bis zwei Personen sinnvoll ist.

Welche Kosten fallen an? DANTE e.V. übernimmt grundsätzlich alle anfallenden Kosten. Die Räumlichkeiten sind an Universitäten oftmals kostenfrei zu bekommen, für die Kaffeepausen steht ein entsprechendes Budget zur Verfügung.

Sollten Sie einen besonderen Tagungsort im Auge haben, der nicht kostenfrei zu haben ist, so stehen wir dem offen gegenüber.

Was ist beim Begleitprogramm zu beachten? Für die Abendtreffs (vier im Frühjahr, zwei im Herbst) sind wir auf den Rat der lokalen Organisatoren angewiesen. Die Restaurants sollten gut zu erreichen sein und kulinarisch wie preislich möglichst »massenkompatibel« sein. Letztlich überlassen wir es aber den lokalen Organisatoren, dies zu entscheiden.

Weitere Ideen wie eine Stadtführung o. ä. sind herzlich willkommen.

An wen soll ich mich bei Fragen oder auch Interesse wenden? Am besten schreiben Sie an den Vorstand (`vorstand@dante.de`).

Bretter, die die Welt bedeuten

Integration von Python in TEX am Beispiel von Katalogeinträgen

Lukas C. Bossert, Uwe Ziegenhagen, Herbert Voß

Viele Dissertationen in der Archäologie enthalten am Ende der Arbeit einen Katalog, in dem die untersuchten Daten in einem bestimmten System aufgeschlüsselt präsentiert werden. In diesem Beitrag wird eine effiziente Gestaltung eines Kataloges und dessen Einbindung in den Fließtext vorgestellt.

Ein Katalog kann aus verschiedenen Einträgen bestehen, wie Bildern, Bohrproben, Architekturelementen etc., die aufgelistet werden. Eine händische Erstellung dieser einzelnen Einträge des Kataloges, beispielsweise über `\section` oder `\subsection` und anschließend in einer Umgebung `itemize`, ist nicht effizient, fehleranfällig und nur bei wenigen Katalogeinträgen einsetzbar. Es muss zudem berücksichtigt werden, dass die vorgegebenen Kategorien nicht für jeden Katalogeintrag passend sind, sodass Kategorien leer bleiben und dann im Katalogeintrag nicht auftauchen sollen. Dabei soll der Code des Katalogeintrags möglichst viel redundante Tipp-Arbeit abnehmen, wie sie beispielsweise bei Maßeinheiten vorkommt. Darüber hinaus sollen alle Einträge immer gleich formatiert sein und ihr Aussehen global verändert werden können.

Nachdem eine Lösung gefunden wurde, die allen bisher genannten Anforderungen entspricht (siehe unten), sollten in einer Kategorie alle Seitenzahlen enthalten sein, auf denen im Haupttext auf den Katalogeintrag verwiesen wird. Ein Lösungsansatz sah die Nutzung von `glossaries` vor: Dafür musste allerdings für jeden Katalogeintrag ein eigenes Glossar angelegt werden, was nicht nur viel händische Arbeit bedeutet, sondern auch die Zählerkapazitäten von TEX überforderte.

Es musste also eine andere Lösung her und die Erinnerung an einen Beitrag in »Die TEXnische Komödie« half. [1] Damals ging es darum, bei einem Werkkatalog die Erwähnung des Stückes im Haupttext anzugeben. Es war genau das, was gesucht wurde und den damals verwendeten Code gab es noch auf `http://uweziegenhagen.de/?p=3020`. Allerdings bestand die damalige Aufgabe darin, nur *eine* Erwähnung

im Haupttext anzugeben, was mit `label` und `ref` bewerkstelligt werden konnte. Im vorliegenden Fall wurden jedoch alle Erwähnungen im Haupttext benötigt. Es wurde entschieden, dieses Problem erst einmal mit der Skriptsprache Python zu lösen.

Um gemeinsam an dem hybriden Konstrukt von XeLaTeX und Python arbeiten zu können, wurde ein Repository auf `github`[1] erstellt, einer Plattform, die unabhängig von der Programmiersprache einen exzellenten Austausch und eine detaillierte Versionskontrolle ermöglicht.

Katalogeintrag

Die Aufgabe war einen Katalog zu den Häusern in Pompeji anzulegen, der Auskunft über den Namen des Hauses, dessen Verortung und Grundstücksgröße geben soll. Zudem soll eine kurze Beschreibung enthalten sein und die Angabe über die Innenausstattung, die wiederum auf die Untergruppen Mosaik, Wandgemälde und Skulptur aufgeschlüsselt werden kann. Zum Schluss soll im Katalogeintrag angezeigt werden, auf welchen Seiten des Fließtextes das Haus genannt wird.

Die gefundene Lösung für die Umsetzung der Anforderungen funktioniert mit Hilfe des Pakets `keyval`.[2] Dafür werden in der Präambel verschiedene Schlüssel (`keys`) definiert. In der Grundversion sieht die Definition eines Eintrags wie folgt aus:[3]

```
\define@key{family}{key}{#1}
```

Für das konkrete Beispiel wird die `key`-Familie (`family`) mit `catalogue` angegeben, der Schlüssel (`key`), was einer Kategorie im Katalog entspricht, als `house` bezeichnet und als Resultat soll der Wert im Makro `\KVhouse` gespeichert werden:

```
\define@key{catalogue}{house}{\def\KVhouse{#1}}
```

Dem Beispiel entsprechend, können alle Kategorien als Schlüssel angelegt werden (Listing 1):

Listing 1: Definition der Schlüssel

```
1 \makeatletter
2 \define@key{catalogue}{house}{\def\KVhouse{#1}}
3 \define@key{catalogue}{label}{\def\KVlabel{#1}}
4 \define@key{catalogue}{description}{\def\KVdescription{#1}}
5 \define@key{catalogue}{location}{\def\KVlocation{#1}}
6 \define@key{catalogue}{size}{\def\KVsize{#1}}
```

[1] `https://github.com/LukasCBossert/DTK-TeX-Python`

[2] Basierend auf der Idee, vorgestellt auf: `http://tex.stackexchange.com/a/254336/98739`

[3] Vgl. `http://www.tug.org/tugboat/tb30-1/tb94wright-keyval.pdf`

```
\define@key{catalogue}{interior}{\def\KVinterior{#1}}
\define@key{catalogue}{interiorM}{\def\KVinteriorM{#1}}
\define@key{catalogue}{interiorW}{\def\KVinteriorW{#1}}
\define@key{catalogue}{interiorS}{\def\KVinteriorS{#1}}
\makeatother
```

Das Aussehen eines Katalogeintrages wird separat mit einem Makro definiert: `\newcommand\catalogueentry[1]`… Darin wird zunächst festgelegt, dass eine neue Gruppe beginnt (`\begingroup`), sodass es zu keinen Problemen mit den jeweils definierten Schlüsseln kommt, da diese für jeden Katalogeintrag neu definiert werden. Dann sollen die Einträge im Flattersatz gesetzt werden (`\RaggedRight`) und schließlich die Angabe, welche Schlüsselfamilie (hier `catalogue`) auszulesen ist.

Listing 2: Definition der Katalogeinträge, Anfang

```
\newcommand\catalogueentry[1]{%
\begingroup
\RaggedRight
\setkeys{catalogue}{#1}
...
```

Es folgt in der Definition die Verarbeitung der einzelnen Schlüssel: Da nur die Ausgabe einer Kategorie erfolgen soll, wenn diese auch mit Informationen versehen ist, wird dies über die Abfrage `\ifdef` erledigt. Der Inhalt der Kategorie `house` wird als `\section`-Titel verwendet und, wenn vorhanden, mit einem `\label` versehen. Die weiteren Kategorien sollen in einer `labeling`-Umgebung aufgelistet werden. Die Definition wird mit `\endgroup` geschlossen.

Listing 3: Definition der Katalogeinträge, Fortsetzung

```
...
\ifdef{\KVhouse}{\section{\KVhouse
      \ifdef{\KVlabel}{\label{\KVlabel}}{}}}{}
\begin{labeling}{Beschreibung}
  \ifdef{\KVdescription}{\item[Beschreibung] \KVdescription}{}
  \ifdef{\KVlocation}{\item[Verortung] \KVlocation}{}
  \ifdef{\KVinterior}{\item[Ausstattung] \KVinterior
      \begin{labeling}{Wandgemälde}
        \ifdef{\KVinteriorM}{\item[Mosaike:] \KVinteriorM}{}
        \ifdef{\KVinteriorW}{\item[Wandgemälde:] \KVinteriorW}{}
        \ifdef{\KVinteriorS}{\item[Statuen:] \KVinteriorS}{}
      \end{labeling}
      }{}
  \ifdef{\KVsize}{\item[Größe] \KVsize\,$\text{m}^2$}{}
\end{labeling}
```

```
\endgroup
}
```

Mit dieser Einstellung lassen sich die Katalogeinträge schon sehr gut darstellen. Im Fließtext des Hauptdokuments kann man an gewünschter Stelle mit `\catalogueentry` einen Katalogeintrag eintragen. So wird aus den folgenden Angaben (Listing 4)

Listing 4: Katalogeintrag für das Haus des M. Fabius Rufus

```
\catalogueentry{%
  house={Haus des M. Fabius Rufus},
  label={haus:M-Fabius-Rufus},
  size={172},
  description={Haus besteht aus mehreren Einzelgebäuden.},
  location={Regio VII, Insula 16, Eingang 17--22.},
  interior={Reicher Fundkomplex.},
  interiorM={S/W-Mosaik.},
  interiorW={Dionysius mit einer Mänade, Narzissus und ein Cupido, Hercules und
 ↪Deinira etc.},
  interiorS={Bronzene Statue eines Epheben.},
}
```

ein Katalogeintrag, der so aussieht:

Haus des M. Fabius Rufus

Beschreibung	Haus besteht aus mehreren Einzelgebäuden.	
Verortung	Regio VII, Insula 16, Eingang 17–22.	
Ausstattung	Reicher Fundkomplex.	
	Mosaike:	S/W-Mosaik.
	Wandgemälde:	Dionysius mit einer Mänade, Narzissus und ein Cupido, Hercules und Deinira etc.
	Statuen:	Bronzene Statue eines Epheben.
Größe	172 m^2	

Wie man bei diesem Beispiel sieht, ist die Reihenfolge, in der die Kategorien angegeben werden, irrelevant, da die Definition in der Präambel entscheidend ist. Der Wert bei `size` wird intern sogleich an das vordefinierte `\SI`-Makro mit entsprechender Einheit (m^2) übergeben. Ähnlich kann auch mit anderen Angaben verfahren werden (bspw. bei Abbildungen können die `\label` an ein vordefiniertes Makro `\cref` übergeben werden).

Bei dem oben beschriebenen Katalogeintrag zum »Haus des M. Fabius Rufus« sind alle Kategorien ausgefüllt. Wenn eine Kategorie nicht ausgefüllt ist, wird sie nicht ausgegeben. Allerdings besteht die Kategorie `interior` [Ausstattung] zusätzlich aus drei Unterkategorien (`interiorM` [Mosaike], `interiorW` [Wandgemälde], `interiorS` [Statuen]). Diese sollen auch dann angezeigt werden, wenn `interior` selbst nicht definiert ist.

Ein solcher Fall tritt beim »Haus des Wilden Ebers« auf. Im Fließtext ist der Katalogeintrag wie folgt ausgefüllt (Listing 5):

Listing 5: Katalogeintrag für das Haus des Wilden Ebers

```
\catalogueentry{%
  house={Haus des Wilden Ebers},
  label={Haus-des-Wilden-Ebers},
  size={54},
  description={Renovierung nach Erdbeben 62\,n.\,Chr.},
  location={Regio VII, Insula 4, Eingang 48, 43.},
  interiorM={S/W-Mosaik.},
  interiorW={Venus, Leda und der Schwan, Ariadne und Theseus.},
}
```

So wird daraus:

Haus des Wilden Ebers

Beschreibung	Renovierung nach Erdbeben 62 n. Chr.	
Verortung	Regio VII, Insula 4, Eingang 48, 43.	
Ausstattung	Mosaike:	S/W-Mosaik.
	Wandgemälde:	Venus, Leda und der Schwan, Ariadne und Theseus.
Größe	54 m^2	

Um zu diesem Ergebnis zu kommen, ist es notwendig, mit Booleschen Operatoren zu arbeiten. Dafür müssen bei den `key`-Definitionen drei Operatoren eingeführt werden (Listing 6):

Listing 6: Definition der Booleschen Operatoren

```
1 \newbool{@KVinteriorM}%Mosaik
2 \newbool{@KVinteriorW}%Wandgemälde
3 \newbool{@KVinteriorS}%Statue
```

Diese Booleschen Operatoren werden bei der Definition der einzelnen Einträge eingebaut und als `true` gesetzt, wenn dieser Eintrag mit Informationen versehen wird. Konkret sieht die Modifikation so aus (Listing 7):

Listing 7: Einsetzten der Booleschen Operatoren in die Schlüssel

```
\define@key{catalogue}{interiorM}{\def\KVinteriorM{#1}\booltrue{@KVinteriorM}}
\define@key{catalogue}{interiorW}{\def\KVinteriorW{#1}\booltrue{@KVinteriorW}}
\define@key{catalogue}{interiorS}{\def\KVinteriorS{#1}\booltrue{@KVinteriorS}}
```

Zudem müssen die Booleschen Operatoren bei der Ausgabe der Katalogeinträge eingesetzt werden, sodass geprüft wird (`\ifboolexpr`), ob einer oder mehrere der Operatoren auf `true` gesetzt ist (Listing 8).

Listing 8: Einsetzten der Booleschen Operatoren in den Katalogeintrag

```
\ifdef{\KVinterior}{%
  \item[Ausstattung] \KVinterior
  \ifboolexpr{bool{@KVinteriorM}
          or bool{@KVinteriorW}
          or bool{@KVinteriorS}}{%
      \begin{labeling}{Wandgemälde}
        \ifdef{\KVinteriorM}{\item[Mosaike] \KVinteriorM}{}
        \ifdef{\KVinteriorW}{\item[Wandgemälde] \KVinteriorW}{}
        \ifdef{\KVinteriorS}{\item[Statuen] \KVinteriorS}{}
      \end{labeling}
      }{}}
  {\ifboolexpr{bool{@KVinteriorM}
          or bool{@KVinteriorW}
          or bool{@KVinteriorS}}{%
    \item[Ausstattung]%
  \begin{labeling}{Wandgemälde}
        \ifdef{\KVinteriorM}{\item[Mosaike] \KVinteriorM}{}
        \ifdef{\KVinteriorW}{\item[Wandgemälde] \KVinteriorW}{}
        \ifdef{\KVinteriorS}{\item[Statuen] \KVinteriorS}{}
      \end{labeling}
  }{}}
```

Mit dieser modifizierten Ergänzung der Katalogeintragsdefinition werden die Unterkategorien von `interior` ausgegeben, auch wenn `interior` selbst nicht definiert ist.

Was noch fehlt, ist die Ausgabe der Seiten, auf denen der Katalogeintrag erwähnt wird. Dies erfolgt zunächst über Python, anschließend wird die TEX-Variante vorgestellt. Lösungen über andere Programmiersprachen sind natürlich auch denkbar.

Problemlösung mit Python

Python ist eine interpretierte Programmiersprache, die den Python-Neuling durch weitestgehende Nicht-Nutzung von Klammern verwirrt. Stattdessen nutzt Python Leerzeichen als Einrückungen, um zusammengehörige Code-Teile zu kennzeichnen. Was erst einmal gewöhnungsbedürftig klingt, hat in der Praxis Vorteile, denn selbst fremder Python-Code ist erstaunlich gut lesbar. Im aktuellen Anwendungsfall beschränken wir uns jedoch nicht auf Python, sondern nutzen mit `pandas` eine Erweiterung zur Datenanalyse.[4] `pandas`, ursprünglich zur Analyse von Finanzdaten entwickelt, erleichtert die Arbeit mit strukturierten Daten aller Art. Das Einlesen, Filtern und Aggregieren von Daten geht damit erstaunlich einfach, dem interessierten Leser sei daher eine nähere Beschäftigung ans Herz gelegt.

Da Python auf Basis des reinen LaTeX-Quelltextes keine Information hat, auf welchen Seiten ein Stichwort erwähnt wird, muss dies auf LaTeX-Ebene weggeschrieben werden.

Dazu bedienen wir uns des Befehls `\newwrite`, der eine neue Ausgabedatei öffnet. In der Präambel des TEX-Dokuments muss aufgeführt werden, dass eine neue Datei geschrieben wird, die den gleichen Namen hat, aber die Endung `.sti`. In diese Datei werden alle Stichwörter und die dazu gehörende Seitenzahl geschrieben, die mit dem Makro `\eintrag{Stichwort}` definiert sind.

```
\newwrite\myfile
\immediate\openout\myfile=\jobname.sti
\newcommand{\eintrag}[1]{\immediate\write\myfile{#1:\thepage}}
```

Für die spätere gesammelte Anzeige der Stichwörter und die entsprechende Seitenzahl müssen zwei weitere Befehle definiert werden:

```
\newcommand{\ausgabe}[2]{#1 was referenced on page(s): #2}
\newcommand{\stichworttabelle}{\IfFileExists{\jobname.tab}{\input{\jobname.tab}}{
 ↪Tab file not found}}
```

Die erste sorgt dafür, dass bei dem Makro `\ausgabe{Stichwort}{ }` das Stichwort sowie der Text *was referenced on page(s):* mit den Seitenzahlen gesetzt wird. Der zweite Befehl gibt die Stichwörter und die Seitenzahlen als Tabelle aus.

Zusammengesetzt sieht der Text in der Präambel wie folgt aus, wobei die Ausgabe des Makros `\ausgabe` zugleich für das konkrete Beispiel abgeändert wurde.

```
1 \newwrite\myfile
2 \immediate\openout\myfile=\jobname.sti
3 \newcommand{\eintrag}[1]{\immediate\write\myfile{#1:\thepage}}
```

[4] `http://pandas.pydata.org`

```
\newcommand{\ausgabe}[2]{#2}
\newcommand{\stichworttabelle}{\IfFileExists{\jobname.tab}{\input{\jobname.tab
 ↪}}{Tab file not found}}
```

Schauen wir uns nun den Python-Code dazu an. Das folgende Listing enthält als Auszug nur die process-Funktion aus der Klasse, den kompletten Quellcode mit einem ausführlich dokumentierten Beispiel findet man im bereits erwähnten Github-Repositorium. Die durch LaTeX erstellte ‹Stichwort:Seite› Datei wird hier über den Befehl read_csv in einen pandas-DataFrame geladen. Diesen DataFrame kann man sich als Tabelle mit zwei Spalten vorstellen, in einer Spalte die Stichwörter, in der anderen die jeweilige Seitenzahl, auf der das Stichwort genannt wurde.

Die Zeile mit »result = df.groupby« enthält recht komplexe Logik. Hier wird nach dem jeweiligen Stichwort gruppiert und ein String erstellt, der die Seitenzahlen, per Komma und Leerzeichen getrennt, enthält. Dabei werden Seiten, auf denen ein Stichwort mehrfach erscheint, auch nur einmal ausgegeben. Was von diesem Befehlt zurückgegeben wird, wird dann wieder in einen pandas-DataFrame verwandelt, um leichter damit arbeiten zu können. Über eine weitere pandas-Funktion wird dieser DataFrame dann in eine LaTeX-Tabelle geschrieben, die dann direkt in das Dokument importiert werden kann (über den selbstdefinierten Befehl \stichworttabelle). Zum Schluss werden in der LaTeX-Datei mittels eines regulären Ausdrucks die \ausgabe-Befehle gesucht und die in den Befehlen vorhandenen Seitenzahlen durch die (unter Umständen) aktuellere Version des Strings im DataFrame ersetzt.

Listing 9: Auszug aus dem Python-Code

```
def process(self,jobname):
 # Lies die Datei im Stichwort:Seite Format
 df = pd.read_csv(jobname+'.sti',sep=':',names=['Stichwort','Seite'])

 # Erstelle die Liste der einzigartigen Stichwörter
 result = df.groupby(['Stichwort'], as_index=False)['Seite'].agg(
       lambda x: ', '.join(np.unique(x).astype(str)))

 # Erstelle einen pandas Dataframe aus der Datenserie
 result = pd.DataFrame(result)

 # setze den Index auf die Spalte 'Stichwort'
 # wird später als Schlüsselfeld genutzt
 result = result.set_index('Stichwort')

 # exportiere den pandas DataFrame in eine LaTeX Tabelle
 result.to_latex(jobname+'.tab',index=True)
```

```
# Gehe durch die Liste der Stichwort/Seiten Tupel und
# suche in der TeX-Datei den \Ausgabe{‹Stichwort›}{‹Seiten›}
# und ersetze den ‹Seiten› Teil durch die aktuelle Version
for index, row in result.iterrows():
 suchmuster = re.compile('(\\\\ausgabe{'+ index + '}{)([0-9, ]*})')
 self.regReplace(jobname+'.tex', suchmuster,
       '\\\\ausgabe{'+ index +'}{' + row[0] + '}')
 print(index, ':', row[0])
```

Integration

Im LaTeX-Dokument muss nun an allen Stellen, an denen das Stichwort auftaucht oder der Rückverweis stattfinden soll, der Befehl `\eintrag{Stichwort}` gesetzt werden. Wenn alle Stichwörter gesetzt sind, kann man die Art der Ausgabe wählen. Entweder werden alle Seitenzahlen zu einem Stichwort angezeigt, dies erfolgt über `\ausgabe{Stichwort}{}` oder Stichwörter und Seiten werden mit `\stichworttabelle` tabellerisch aufgelistet.

Für unser konkretes Beispiel der Katalogeinträge bedeutet dies, dass das Makro `\ausgabe` in die Katalogeintragsdefinition integriert werden muss.

```
\define@key{catalogue}{backref}{\def\KVbackref{#1}}
```

Damit wird der Wert, der im `\catalogueentry` unter `backref` angegeben ist (hier *Haus*), an den Befehl `\ausgabe` übergeben:

```
\catalogueentry{%
  backref={\ausgabe{Haus}{}},
}
```

Das bedeutet selbstverständlich, dass im Fließtext entsprechend `\eintrag{Haus}` gesetzt werden muss. Wenn etwas im `backref` des Katalogeintrags steht, dann wird dieser Eintrag übernommen, ansonsten wird geschaut, ob das `label` definiert ist, wenn auch dies nicht der Fall ist, bleibt dieser Katalogeintrag undefiniert und es findet auch keine Weiterverarbeitung statt.

Für die Darstellung des Katalogeintrages muss ebenfalls eine Neudefinition eingeführt werden:

```
\ifdef{\KVbackref}{\item[Erwähnungen] \KVbackref}{}
```

Ein vollständiger Katalogeintrag:

```
\catalogueentry{%
  house={Haus des M. Fabius Rufus},
```

```
  label={haus:M-Fabius-Rufus},
  size={172},
  description={Haus besteht aus mehreren Einzelgebäuden.},
  location={Regio VII, Insula 16, Eingang 17--22.},
  interior={Reicher Fundkomplex.},
  backref={\ausgabe{Haus-Fabius}{}},
}
```

Nun wird *Haus* an `\ausgabe` übergeben. Im folgenden Fall ist dies der Wert vom Katalogeintrag `label`:

```
\catalogueentry{%
  house={Haus des Wilden Ebers},
  label={Haus-des-Wilden-Ebers},
  size={54},
  description={Renovierung nach Erdbeben 62\,n.\,Chr.},
  location={Regio VII, Insula 4, Eingang 48, 43.},
}
```

Während im ersten Fall im Fließtext an gewünschter Stelle `\eintrag{Haus-Fabius}` gesetzt werden muss, gilt für die zweite Variante `\eintrag{Haus-des-Wilden-Ebers}`.

Haus des M. Fabius Rufus

Beschreibung	Haus besteht aus mehreren Einzelgebäuden.
Verortung	Regio VII, Insula 16, Eingang 17–22.
Ausstattung	Reicher Fundkomplex.
Größe	172 m^2
Erwähnungen	S.

Haus des Wilden Ebers

Beschreibung	Renovierung nach Erdbeben 62 n. Chr.
Verortung	Regio VII, Insula 4, Eingang 48, 43.
Größe	54 m^2
Erwähnungen	S.

Sobald diese Vorarbeiten getan sind, alle Stichwörter entsprechend mit `\eintrag` gesetzt sind, dann muss einmal (bspw. mit XeLaTeX) kompiliert werden, anschließend

der Python-Code im entsprechenden externen Programm ausgeführt und schließlich zurück im TEX-Editor wiederum einmal kompiliert werden.

Problemlösung mit TEX

Bei einer Lösung über TEX muss beachtet werden, dass die jeweiligen Seitenzahlen der Referenzen eventuell schon benötigt werden, wenn der eigentliche Referenzbefehl noch gar nicht ausgeführt wurde. Dies entspricht faktisch dem gleichen Vorgang wie bei der Erstellung eines Inhaltsverzeichnisses. Da dieses im Allgemeinen am Anfang des Dokumentes steht, kann es auch erst nach einem zweiten Durchlauf ausgegeben werden. Die gleiche Vorgehensweise wird für die Verweise gewählt:

- Erstellen einer eigenen externen Referenzdatei.
- Beim Programmstart Einlesen dieser Datei, sofern sie von einem vorhergehenden Lauf vorhanden ist.
- Interpretieren der in der Datei vorhandenen Referenzbefehle.
- Löschen der Datei und erneutes Schreiben der Referenzen in diese, um Änderungen zu erfassen.
- Am Ende schließen und speichern der externen Datei.

Die externe Datei bekommt den Namen `\jobname.ctg`:

```
\newwrite\@catalog%   Neues write-Register
\InputIfFileExists{\jobname.ctg}{}{}% Datei einlesen, falls schon vorhanden
\immediate\openout\@catalog\jobname.ctg% Zum Überschreiben öffnen
```

Das heißt, dass man im Gegensatz zur Python-Variante die Rückverweise nicht manuell setzen muss, sondern sie werden automatisch über die im Fließtext gesetzten `\labelcref` erstellt. Ein entsprechender Eintrag sieht dann wie folgt aus:

```
\catalogueentry{%
  house=Haus des Wilden Ebers,
  label=haus:Haus-des-Wilden-Ebers,
  size=54,
  description={Renovierung nach Erdbeben 62\,n.\,Chr.},
  location={Regio VII, Insula 4, Eingang 48, 43},
  interiorM=S/W-Mosaik,
  interiorW={Venus, Leda und der Schwan, Ariadne und Theseus},
  mark%  Gib Seitenzahlen aus
}
```

Allerdings muss das Makro `\labelcref` modifiziert werden, sodass es zusätzlich die entsprechenden Einträge in die externe Datei schreiben kann:

```
\let\LabelCref\labelcref
\renewcommand\labelcref[1]{\expandafter\label@cref#1,,\@nil}
\def\label@cref#1,#2,#3\@nil{%
  \immediate\write\@catalog{\string\catalog@ref{#1-pages}{\arabic{page}}}
  \LabelCref{#1}%
  \ifx\relax#2\relax % kein label mehr da
    \def\next{}%
  \else
    \def\next{\label@cref#2,#3\@nil}%
  \fi
  \next}
\AtEndDocument{\closeout\@catalog}
```

Die Datei `\jobname.ctg` könnte dann beispielsweise folgende Einträge enthalten:

```
\catalog@ref{haus:Haus-des-Wilden-Ebers-pages}{2}
\catalog@ref{haus:M-Fabius-Rufus-pages}{3}
\catalog@ref{haus:Haus-des-Wilden-Ebers-pages}{3}
\catalog@ref{haus:M-Fabius-Rufus-pages}{4}
\catalog@ref{haus:Haus-der-Venus-pages}{4}
\catalog@ref{haus:Haus-des-Wilden-Ebers-pages}{4}
\catalog@ref{haus:Haus-der-Venus-pages}{5}
```

Diese werden durch die entsprechende Definition von `\catalog@ref` interpretiert:

```
\def\catalog@ref#1#2{%
  \expandafter\ifx\csname#1\endcsname\relax  % schon definiert??
    \@namedef{#1}{#2}% Nein, also erstellen
  \else% anderenfalls Seitenzahl an existierenden Befehl anhängen
    \expandafter\edef\csname#1\endcsname{\csname#1\endcsname, #2}%
  \fi}
```

Es existieren danach jeweils Makros, die für die zugehörigen Seitenzahlen stehen. Beispielsweise enthält das Makro `\haus:Haus-des-Wilden-Ebers-pages` als Wert die Seitenangaben `2, 3, 4`. Auf diese Makros muss dann einfach bei den Katalogeinträgen Bezug genommen werden:

```
\newcommand\catalogueentry[1]{%
...
  \ifKVmark \item[Erwähnungen] S.~\expandafter\csname\KVlabel-pages\endcsname \fi
...
}
```

Die Seiten, auf denen über ein `\labelcref` auf ein Katalogeintrag verwiesen wird, werden nun beim jeweiligen Katalogeintrag eingetragen und bei jedem *zweiten*

Kompilieren auf den aktuellen Stand gebracht, insofern sich zwischenzeitlich etwas geändert hat. Als Beispiel sei die erste Seite eines Dokumentes ausgegeben, welches alle Verweise grundsätzlich erst nach der Definition der einzelnen Katalogeinträge vornimmt.

Zusammenfassung

Ausgehend von dem Wunsch, einen Katalog in einer wissenschaftlichen Arbeit möglichst effizient zu gestalten, wurde die Lösung über die Definition verschiedener keys präsentiert, die eine einheitliche Programmierung der Eingabe und Gestaltung der Ausgabe ermöglichen.

Zwei Möglichkeiten wurden vorgestellt, wie man einen Rückverweis im jeweiligen Katalogteil auf die Seite mit der Erwähnung einbauen kann. Die erste Lösung arbeitet mit einem Python-Code, bei dem man den Marker `\eintrag{Stichwort}` im Text selbst setzen kann und bei dem die Seitenzahl über `\ausgabe{Stichwort}{}` im Katalogeintrag ausgegeben wird.

Die zweite Möglichkeit arbeitet mit TEX-eigenen Mitteln, indem es die `\labelcref` nutzt, mit denen auf die Katalogeinträge verwiesen wird. Diese Variante erfordert kein händisches Setzen der Marker im Text, führt dann jedoch nur die Seiten auf, auf denen auch ein entsprechendes `\labelcref` gesetzt ist.

Für beide Lösungen wurde ein Minimalbeispiel erstellt, sodass man diese Varianten ausprobieren kann.[5]

Literatur und Software

[1] Uwe Ziegenhagen: »Größere Dokumente mit LATEX erstellen«, *Die TEXnische Komödie*, *1* (Feb. 2015), 25–29.

[5] Für den Python-Code siehe https://github.com/LukasCBossert/DTK-TeX-Python/blob/master/catalogueentry-python.tex, für den TEX-Code siehe https://github.com/LukasCBossert/DTK-TeX-Python/blob/master/catalogueentry-tex.tex.

1 Katalogteil

Kat. A.1 Haus des M. Fabius Rufus

Beschreibung	Haus besteht aus mehreren Einzelgebäuden.	
Verortung	Regio VII, Insula 16, Eingang 17–22.	
Ausstattung	Reicher Fundkomplex.	
	Mosaike:	S/W-Mosaik
	Wandgemälde:	Dionysius mit einer Mänade, Narzissus und ein Cupido, Hercules und Deinira, etc.
	Statuen:	Bronzene Statue eines Epheben
Größe	172 m^2	
Erwähnung	S. 2, 3	

Kat. A.2 Haus des Wilden Ebers

Beschreibung	Renovierung nach Erdbeben 62 n. Chr.	
Verortung	Regio VII, Insula 4, Eingang 48, 43	
Ausstattung	Mosaike:	S/W-Mosaik
	Wandgemälde:	Venus, Leda und der Schwan, Ariadne und Theseus
Größe	54 m^2	
Erwähnung	S. 1, 2, 3	

Kat. A.3 Haus der Venus

Beschreibung	Schönes großes Haus
Verortung	Regio VII, Insula 4, Eingang 12, 23
Ausstattung	Keine Innenausstattung
Größe	123 m^2
Erwähnung	S. 3, 4

2 Text

In meiner Arbeit beschäftige ich mit drei Häusern in Pompeji (Kat. A.2).

Parallel TEXen mit Python

Uwe Ziegenhagen

In diesem Artikel möchte ich kurz vorstellen, wie mit Unterstützung durch ein Python-Skript die in vielen Rechnern mehrfachen vorhandenen CPU-Kerne dazu benutzt werden können, LATEX-Läufe zu parallelisieren, um so Zeit bei der Übersetzung zu sparen.

Einführung

Der durchschnittliche Computer-Nutzer hat heutzutage auf dem Schreibtisch weit mehr Rechenleistung, als der NASA für den Flug zum Mond bereitstand[1]. Mehrere CPU-Kerne oder sogar mehrere CPUs sowie eine vor 20 Jahren unvorstellbar große Menge RAM »langweilen« sich die meiste Zeit, weil in vielen Fällen der Mensch vor dem Rechner der limitierende Faktor ist. So schnell, wie ein moderner Rechner rechnen kann, kann in den meisten Fällen der Mensch nicht die Aufgaben heranschaffen, die es zu bearbeiten gilt.

Beim Verarbeiten von umfangreichen TEX-Dokumenten ist es jedoch auch heute noch so, dass auch moderne Rechner einige Minuten brauchen, um beispielsweise die hunderten Seiten einer Dissertation mehrfach zu übersetzen.

Ausgangspunkt dieses Artikels war die Frage, ob nicht durch *parallele* LATEX-Läufe ein Geschwindigkeitszuwachs erzielt werden kann. Als technische Plattform wird Python 3 genutzt, andere Programmiersprachen und selbst `make`[2] halten jedoch ähnliche Funktionalitäten bereit.

Python-Code

In diesem Abschnitt möchte ich kurz auf den Beispiel-Code in Listing 1 eingehen, den ich basierend auf einem Beispiel aus Stackoverflow[3] für LATEX angepasst habe. Es werden drei Python-Bibliotheken geladen: `multiprocessing`, `time` und `os.multiprocessing` kümmern sich um die Parallisierung der Aufgaben, `time` stellt eine Stoppuhr bereit, um den Leistungszuwachs messen zu können und `os` setzt die Betriebssystembefehle zum Aufruf von `pdflatex` ab.

Die Variable `files` bekommt eine Liste von zehn identischen TEX-Dateien übergeben, die dann parallel übersetzt werden sollen. Die einzelnen Dateien enthalten nur einige

[1] `http://www.computerweekly.com/feature/Apollo-11-The-computers-that-put-man-on-the-moon`
[2] `www.gnu.org/software/make/manual/html_node/Parallel.html`
[3] `http://stackoverflow.com/questions/16181121/python-very-simple-multithreading-parallel-url-fetching-without-queue`

übliche Pakete sowie einige durch das Paket `blindtext` erstellte Absätze. Auf eine Darstellung kann daher hier verzichtet werden. `compileFile()` ist die Funktion, die den Aufruf von `pdflatex` für ein übergebenes Dokument (in der Variable `cfile` gespeichert) steuert. Die Funktion versucht, die übergebene Datei im Batch-Mode (also mit möglichst wenig Ausgabe) zu übersetzen, bei auftretenden Fehlern wird eine Exception ausgelöst und der Exception-Text zurückgegeben.

`start = timer()` startet die Stoppuhr, auf die dann bei den einzelnen Läufen referenziert wird. Die folgende Zeile sorgt dafür, dass der `ThreadPool` erstellt wird, dem dann die Aufgabe sowie die Liste der zu übersetzenden Dateien übergeben wird. Einen `ThreadPool` muss man sich als Sammlung von Arbeitsbienen vorstellen, in diesem Fall acht Stück,was der der Zahl der CPU-Kerne in meinem Rechner und damit der Zahl der maximal *gleichzeitig* parallelisierbaren Aufgaben entspricht. Der Rest des Codes sorgt nur dafür, dass die gesetzte Zeit – beziehungsweise im Fehlerfall die Fehlermeldung – ausgegeben wird.

Listing 1: Quellcode für den ersten Test

```
from multiprocessing.pool import ThreadPool
from time import time as timer
import os

files = ['test-01.tex', 'test-02.tex', 'test-03.tex', 'test-04.tex',
'test-05.tex', 'test-06.tex', 'test-07.tex', 'test-08.tex',
'test-09.tex', 'test-10.tex']

def compileFile(cfile):
  try:
    result = os.system('pdflatex -interaction=batchmode ' + cfile)
    return cfile, None
  except Exception as e:
    return cfile, e

start = timer()
results = ThreadPool(8).imap_unordered(compileFile, files)
for cfile, error in results:
  if error is None:
    print("%r compiled in %ss" % (cfile, timer() - start))
  else:
    print("Error compiling %r: %s" % (cfile, error))
    print("Elapsed Time: %s" % (timer() - start,))

print('Gesamtzeit',timer() - start)
```

Parallele Spendenbescheinigungen

Während die Unterschiede im oben erwähnten Beispiel zwischen rein sequentieller und paralleler Bearbeitung nur bei wenigen Sekunden (7,9 Sekunden versus 3,6 Sekunden) liegen, bringt das nächste Beispiel deutlichere Unterschiede. In DTK 2/2014 hatte ich gezeigt, wie man mit Hilfe von LaTeX, MySQL und Python Spendenbescheinigungen setzen kann, der Prozess generiert dabei für jede Spendenbescheinigung eine eigene TEX-Datei.

Für das Jahr 2015 hatte ich in meiner Funktion als Schatzmeister des Kölner Dingfabrik e.V. 88 einzelne Dokumente, also eine gute Basis für einen Test. Als Rechner stand ein Dual-Xeon mit insgesamt acht Kernen, 48 GB RAM und installiertem TEX Live 2016 zur Verfügung, alle TEX-Dateien befanden sich in einer RAM-Disk. Sequentiell dauerte die Übersetzung knapp 60 Sekunden, parallelisiert weniger als 8 Sekunden. Ein weiterer Test mit diesen Dateien auf einem i7 unter Windows 8 – mit den Dateien auf einer SSD – ergab rund 74 Sekunden für die sequentielle Übersetzung und ungefähr 24 Sekunden für die parallelisierte Übersetzung.

Fazit

Im alltäglichen Gebrauch ist es vermutlich nicht wirklich notwendig, TEX-Dateien parallel zu übersetzen, in den meisten Fällen ist die sequentielle Verarbeitung sogar notwendig, um die Abhängigkeiten zu `makeindex` und `biblatex/biber` aufzulösen.

Bei einer Vielzahl ähnlicher Dokumente, die es zu übersetzen gilt, zeigen sich aber signifikante Geschwindigkeitszuwächse. Darüber hinaus macht es natürlich Spaß, die vorhandene Hardware richtig ausnutzen zu können…

Wie immer freue ich mich über Anregungen und Ideen.

Trennmuster und deren Anwendung

Herbert Voß

Da sich die Trennregeln in fast allen Sprachen einer algorithmischen Beschreibung entziehen, kann man nur auf Datenbanken oder wahrscheinlichkeitsbasierte Verfahren zurückgreifen. Im Allgemeinen kümmert man sich nicht um die internen Vorgänge bei der Ermittlung der möglichen Trennungen. In manchen Fällen möchte man aber wissen, warum ein Wort nicht anders getrennt wurde, beziehungsweise welche Trennungen überhaupt möglich sind.

Datenbank versus Trennmuster

Der Vorteil einer Datenbank mit der Zuordnung von Wörter ohne Trennungen zu den korrespondierenden mit Trennungen (»Trennungen ↔Tren-nun-gen«) ist die fehlerfreie Anwendung. Der Nachteil ist, dass unbekannte Wörter nicht getrennt werden können. Letzteres ist dagegen bei dem wahrscheinlichkeitsbasierten Verfahren, wie TEX es verwendet, möglich. Der Nachteil ist eindeutig, dass es zu fehlenden oder fehlerhaften Trennungen kommen kann. Fehlerhafte Trennungen sind dabei zwar selten – weniger als jede 1000te Worttrennung ist falsch – aber dennoch ärgerlich. [6] Trotzdem überwiegt der Vorteil, sodass das Verfahren, wie es von TEX angewendet wird, momentan als sinnvollste Variante erscheint.

Der Trennalgorithmus

Grundlage für die sogenannten Trennmuster ist *immer* eine Datei mit korrekten Trennungen und das Ergebnis *immer* eine Datei mit gewichteten Trennmustern. Als Beispiel wird hier eine Basis von 13 Wörtern gewählt. Mit dem Programm `patgen` (pattern generation) lassen sich daraus die angegebenen Trennmuster erstellen. Eine Programmkurzbeschreibung bekommt man mit `texdoc patgen`, eine etwas längere Einführung liegt auf CTAN: `http://mirrors.ctan.org/info/patgen2/patgen2.pdf` und die Dissertation von Frank Liang[1] aus dem Jahr 1983 liegt auf der TUG-Webseite: `https://tug.org/docs/liang/liang-thesis.pdf`. Das Repositorium der »experimentellen« deutschen Trennmuster, die aufgrund einer erheblich größeren Datenbasis erstellt werden, findet man auf `http://repo.or.cz/wortliste.git`. Ihre Verwendung in LATEX-Dokumenten wurde in [4] beschrieben. Falsche oder nicht gefundene Trennungen werden in einer Ausnahmedatei erfasst. [3, 2]

[1] Frank Liang: »Word Hy-phen-a-tion by Com-put-er«. Eine verbesserte Version des Programms PatGen wurde 1991 von Peter Breitenlohner erstellt.

Hier soll nur zur Demonstration ein kleines Beispiel angegeben werden. Wegen der sehr kleinen Datenbasis ist die Wahrscheinlichkeit richtiger Trennungen klein und es gibt bis auf `r2st` nur Zweiermuster.

```
je-na
trenn-re-gel
for-schungs-kol-lo-qui-um
kol-lo-qui-um
u-ni-ver-si-tät
fried-rich
schil-ler
ger-ma-nis-tik
in-s-ti-tut
für
ger-ma-nis-ti-sche
sprach-wis-sen-schaft
fürs-ten-gra-ben
```

⇒

```
a1b  a1n  d1r  e1g
1gr  h1w  i1t  i1u
i1v  l1l  1na  1ni
n1r  n1s  o1q  r1m
r1s  r2st 1sc  s1k
s1s  s1t
```

Die Muster sind so aufgebaut, dass sie positive Wahrscheinlichkeiten (Trennung möglich) durch die ungeraden Ziffern 1 … 9 und negative Wahrscheinlichkeiten (Trennung nicht möglich) durch die geraden Ziffern 2 … 8 angeben. Je größer die Ziffer, desto größer die jeweilige Wahrscheinlichkeit. Zum Vergleich sei ein Auszug aus den aktuellen deutschen Trennmuster angegeben:

```
ca5y c1c 1ce celi4c celich5 ce1ro c8h 2ch.  1chae ch1ah ch3akt cha6mer
8chanz 5chara 3chari 5chato 6chb 1chef 6chei ch3eil ch3eis 6cherkl
6chf 4chh 5chiad 5chias 6chins 8chj chl6 5chlor 6ch2m 2chn6 ch8nie
```

Grundsätzlich gilt, dass die Wörter, deren Trennungen Teil der Datenbasis waren, richtig getrennt vorliegen müssen. Das zu trennende Wort wird dann auf TEX-Ebene zuerst in Kleinbuchstaben gewandelt, dann in Punkte eingeschlossen und abschließend in überlappende Zweier-, Dreier-, … Gruppen zerlegt. Diesen Gruppen werden vorhandene Trennmuster zugeordnet.

Trennbeispiele

Zerlegung des Wortes »Universität«:

```
.universität.
.u un ni iv ve er rs si it tä ät t.
.un niv ver rsi itä tät ät.
.uni iver rsit tät.
.univ versi ität.
...
```

Anwendung der obigen Trennmuster:

```
. u n i v e r s i t ä t .
  1n i
     i1v
          r1s
              i1t
  u1n i1v e r1s i1t ä t
  u-n i-v e r-s i-t ä t   -> U-ni-ver-si-tät
```

Da das Wort »Universität« Teil der Datenbasis war, wird es auch richtig getrennt. Obwohl die Datenbasis zufällig gewählt wurde und nur 13 Wörter enthält, kann sie ohne weiteres auf unbekannte Wörter angewendet werden. Zerlegung des Wortes »Krankenschwester«:

```
.krankenschwester.
.k kr ra an nk ke en ns sc ch hw we es st te er r.
.kr ran nke ens sch hwe est ter.
.kra anke ensc chwe este r.
...
```

Anwendung der Trennmuster:

```
. k r a n k e n s c h w e s t e r .
      a1n     n1s
      a1n      1sc        s1t
  k r a1n k e n1s c h w e s1t e r  -> Kra-nken-schwes-ter
```

Das der Datenbank unbekannte Wort »Krankenschwester« enthält eine falsche und zwei richtige Trennungen. Eine Kontrolle mit den aktuellen Trennmustern von TEXLive 2016 ergibt:

```
voss@shania:~> mtxrun --script patterns --language=de --hyphenate Krankenschwester

resolvers    | trees | analyzing 'home:texmf'
hyphenator   |
hyphenator   | . k r a n k e n s c h w e s t e r .   . k r a n k e n s c h w e s t e r .
hyphenator   |  0k2r4                                 0 2 4 0 0 0 0 0 0 0 0 0 0 0 0 0
hyphenator   |        2n1k0                           0 2 4 2 1 0 0 0 0 0 0 0 0 0 0 0
hyphenator   |              6n1s0                     0 2 4 2 1 0 6 1 0 0 0 0 0 0 0 0
hyphenator   |                1s0c0                   0 2 4 2 1 0 6 1 0 0 0 0 0 0 0 0
hyphenator   |                  0c4h0                 0 2 4 2 1 0 6 1 0 4 0 0 0 0 0 0
hyphenator   |                  4c0h0w0               0 2 4 2 1 0 6 1 4 4 0 0 0 0 0 0
hyphenator   |                    2h0w0               0 2 4 2 1 0 6 1 4 4 0 0 0 0 0 0
hyphenator   |                       0w0e4s0t0        0 2 4 2 1 0 6 1 4 4 0 0 4 0 0 0
hyphenator   |                            0s1t0       0 2 4 2 1 0 6 1 4 4 0 0 4 1 0 0
hyphenator   |                            2s0t0e0r0   0 2 4 2 1 0 6 1 4 4 0 0 4 1 0 0
hyphenator   |                            6s5t0e0r0.0 0 2 4 2 1 0 6 1 4 4 0 0 6 5 0 0
hyphenator   |                              1t0e0     0 2 4 2 1 0 6 1 4 4 0 0 6 5 0 0
hyphenator   |                              3t6e0r0.0 0 2 4 2 1 0 6 1 4 4 0 0 6 5 6 0
hyphenator   | .0k2r4a2n1k0e6n1s4c4h0w0e6s5t6e0r0.   . k r a n-k e n-s c h w e s-t e r .
hyphenator   |
mtx-patterns | de 3 3 : Krankenschwester : Kran-ken-schwes-ter
```

Das Lua-Script `mtxrun` wird von ConTEXt zur Verfügung gestellt und kann auf der Kommandozeilenebene (Terminal) unter allen Betriebssystemen aufgerufen werden. Die Angabe »`de 3 3`« kennzeichnet zum einen die Sprache (Deutsch) und zum anderen die links- beziehungsweise rechtsseitige Mindestanzahl an Buchstaben. Dies entspricht den Vorgaben von `\lefthyphenmin` und `\righthyphenmin`, wie sie in LATEX gelten. Die Werte können für `mtxrun` durch die Optionen `--left=‹Zahl›` und `--right=‹Zahl›` geändert werden.

Innerhalb eines LATEX-Laufes lassen sich die Trennstellen in der Logdatei durch das Makro `\showhyphens` ausgeben. Das Testprogramm

```
[...]
\usepackage[ngerman]{babel}

\begin{document}
  \showhyphens{Krankenschwester}
\end{document}
```

führt zu folgender Ausgabe in der Logdatei:

```
[...]
Underfull \hbox (badness 10000) in paragraph at lines 8--8
[] \T1/cmr/m/n/10.95 Kran-ken-schwes-ter
[...]
```

Mit dem Paket `showhyphens` von Patrick Gundlach kann man sich für LuaLATEX alle vom Algorithmus gefundenen Trennstellen visuell ausgegeben lassen, wie dies im folgenden Abschnitt durch die dünnen senkrechten Linien getan wird.

Erreicht wird dies auf der unteren Ebene von LuaTEX durch einen sogenannten `callback`. In diesem Fall »`post_linebreak_filter`«, welcher nach einem von TEX ermittelten Zeilenumbruch nachträglich senkrechte Linien an den von TEX ermittelten möglichen Trennstellen einfügt. Dabei ist darauf zu achten, dass bei fremdsprachlichen Textteilen die Sprache entsprechend gewechselt wird, beispielsweise mit `\foreignlanguage{english}{English text}`.

Eine Ausgabe der Wörter mit ihren Trennstellen auf der Ebene von pdfLATEX wurde in [5] gezeigt. Den dort angegebenen Quellcode gibt es auf `https://www.dante.de/DTK/Software/Trennen.tex`.

Die wahrscheinlichkeitsbasierten Trennungen lassen sich durch folgende Eingriffe beeinflussen, wobei die Aufzählung eine absteigende Priorität hat:

1. Individuelle Vorgabe von Trennstellen im Text: `Trenn\-stellen` wird bei `\-` getrennt oder gar nicht.

2. Dokumentenweite Vorgabe durch `\hyphenation{Trenn-stellen}`, was in der Regel in der Präambel erfolgt.
3. Angabe in der Ausnahmeliste (hyphenation exception log) `dehyphtex.tex`, die im Prinzip aber nur Trennvorgaben enthalten sollte, die ansonsten falsch oder ungünstig erfolgen. Beispielsweise würde »letzthin« gar nicht und »injizierbar« nur als »inji-zier-bar« getrennt werden. Die Ausnahmedatei definiert daher »letzt-hin« und »in-ji-zier-bar«. [3]

Zu beachten ist, dass LuaLaTeX die neuen, formal noch experimentellen Trennmuster benutzt, pdfLaTeX dagegen die standardmäßigen, welche eine weitaus kleinere Datenbasis haben.

Literatur und Software

[1] DANTE e.V. : Trennmuster-Wiki, 2016, `http://projekte.dante.de/Trennmuster` (besucht am 04. 08. 2016).

[2] Stephan Hennig: »Einige Fragen zum Beitrag ›Hyphenation Exception Log für deutsche Trennmuster, Version 1‹«, *Die TeXnische Komödie*, *1/2008* (Jan. 2008), 7–17.

[3] Werner Lemberg: »Hyphenation Exception Log für deutsche Trennmuster, Version 1«, *Die TeXnische Komödie*, *2/05* (Mai 2005), 24–51.

[4] Rolf Niepraschk: »Die experimentellen Trennmuster für LaTeX«, *Die TeXnische Komödie*, *3/2011* (Aug. 2011), 49–52.

[5] Heiko Oberdiek, Christine Römer: »Anzeigen der Trennstellen«, *Die TeXnische Komödie*, *3/2010* (Aug. 2010), 16–17.

[6] Christine Römer, Herbert Voß: »Deutsche Silbentrennmuster – aus linguistischer und TeXnischer Sicht«, *Deutsche Sprache*, 3 (2010), 257–282.

Von fremden Bühnen

Im Netz gefunden

Herbert Voß

> In den verschiedenen Mailinglisten, Webforen, Newsgroups u. a. findet man immer wieder hilfreiche Angaben zur Arbeit mit und um das Thema Textsatz mit TEX, LATEX, ConTEXt usw.

Wie liest TEX Quellcode[1]

Zur Erklärung möchte ich etwas weiter ausholen und erstmal versuchen, grob darzulegen, wie TEX Quelltext einliest und Token erzeugt. Man kann sich behelfsmäßig folgendes vorstellen:

- TEX sei eine Fabrik.
- Am Anfang der Produktionsstraße dieser Fabrik sitze ein kleines Männchen.
- Dieses Männchen habe ein Blatt Papier in der Hand und liest dieses Papier. Das Papier entspreche der TEX-Datei.
- Vor dem Männchen sei ein Fließband aufgebaut.
- Neben dem Männchen sei ein Regal.
- In diesem Regal seien viele hübsche glitzernde seltsame faszinierende Dinge einsortiert: Sogenannte Token.

Da gibt es Kontrollsequenz-Token, die im Regal durch Fächer weiter unterteilt werden in Kontrollwort-Token und Kontrollsymbol-Token. Da gibt es Character-Token aller Arten, sortiert nach Category-Code und Character-Code.

Das Männchen liest also das Papier. Und je nachdem, was es dort gerade liest, greift es ins Regal neben sich, nimmt dort Token heraus und legt die herausgenommenen Token aufs Fließband vor sich. Die werden dann auf diesem Fließband der TEX-Fabrik weitertransportiert und weiterverarbeitet.

Das Papier mit dem Quelltext stellt für das Männchen also einen Satz Instruktionen dar, aus dem es erfährt, welche Token es der Reihe nach aus dem Regal neben sich nehmen und aufs Fließband vor sich legen soll.

[1] Ulrich Diez am 22.11.2014 in `news://de.comp.text.tex`

Wie die aufs Fließband gelegten Token weiterverarbeitet werden, ist unterschiedlich. Manche sind expandibel, andere sind Assignments. Wieder andere werden zu Angaben darüber, welches Zeichen aus welcher Schrift in die Ausgabedatei (dvi/pdf) gepackt werden soll. Da gibt es vieles. Aber für den Moment sei das uninteressant. Im Moment interessiert nur der erste Schritt: Das Einlesen und Tokenizen.

Das Männchen liest also die TEX-Datei, beziehungsweise das Blatt Papier und nimmt entsprechend Token aus seinem Regal und legt sie aufs Fließband.

Wie das Männchen beim Lesen die Anweisungen interpretiert und welche Token es folglich aufs Fließband legt, ist abhängig von diversen Aspekten und Regeln.

Ein solcher Aspekt wären zum Beispiel die Category-Codes, die einzelnen Zeichen zugeordnet sind. Kann man sich vorstellen wie eine Tabelle, die vom Management der TEX-Fabrik herausgegeben wird, in der das Männchen nachschlägt, welcher Category-Code einem gerade erblickten Zeichen zugeordnet ist.

Ein anderer solcher Aspekt wäre zum Beispiel der »Status des Leseapparats«. Da kann man sich ein Statusboard vorstellen, welches immer genau eine der folgenden drei möglichen Meldungen anzeigt:

- Erste mögliche Meldung: »Leerzeichen sind zu ignorieren!«
- Zweite mögliche Meldung: »Es wird in einer Zeile der Eingabe gelesen!«
- Dritte mögliche Meldung: »Neue Zeile einlesen!«

Beispiele für Regeln wären:

- Wenn das Männchen auf dem Blatt Papier/in der Eingabe einen Backslash oder ein anderes Zeichen mit Category-Code 0 (Escape Character) erblickt, weiß das Männchen, dass jetzt kein Character-Token, sondern ein Kontrollsequenz-Token aufs Fließband gelegt werden soll.
- Wenn das Männchen auf dem Blatt Papier/in der Eingabe ein Zeichen erblickt, das laut Catcode-Tabelle den Category-Code 5 (end of line; normalerweise ist das bei <return> beziehungsweise ASCII-13 der Fall) hat, wird das Männchen nachschauen, welchen Status der Leseapparat gerade hat.
  - Wenn der Leseapparat gerade die Meldung »Leerzeichen sind zu ignorieren!« anzeigt, wird es keinerlei Token aufs Fließband legen.
  - Wenn der Leseapparat gerade die Meldung »Es wird in einer Zeile der Eingabe gelesen!« anzeigt, wird es ein Space-Token aufs Fließband legen.
  - Wenn der Leseapparat gerade die Meldung »Neue Zeile einlesen!« anzeigt, wird es ein Kontrollwort-Token `\par` aufs Fließband legen.

  Dann wird es sich nicht weiter mit der Eingabezeile befassen, in der das Category-Code-5-Zeichen steht und so gesehen alles ignorieren, was hinter dem Category-Code-5-Zeichen noch in derselben Zeile steht . (Wenn es die nächste

Zeile betrachtet, wird es dann zunächst den Status des Leseapparats auf »Neue Zeile einlesen!« umstellen.)
- Wenn das Männchen auf dem Blatt Papier/in der Eingabe ein Zeichen erblickt, das laut Catcode-Tabelle den Category-Code 14 (comment; normalerweise `%`) hat, wird das Männchen kein Token aufs Fließband legen sondern anfangen, die nächste Zeile der Eingabedatei zu lesen (wobei dann zunächst der Status des Leseapparats auf »Neue Zeile einlesen!« gestellt wird). Somit wird es alles ignorieren, was hinter dem `%` noch in der selben Zeile steht.
- Wenn das Männchen gerade ein Space-Token oder ein Kontrollwort-Token aufs Fließband gelegt hat, stellt es den Status des Leseapparates auf »Leerzeichen sind zu ignorieren!«. (Bei Kontrollsymbol-Token, die auch Kontrollsequenz-Token sind, macht es das nicht, sondern stellt auf »Es wird in einer Zeile der Eingabe gelesen!«.)

Das Männchen liest übrigens zeilenweise. Leerzeichen am Anfang einer Zeile ignoriert es immer. Am Ende jeder Zeile fügt es ein Zeichen an, dessen Character-Code dem Wert des Integer-Parameter `\endlinechar` entspricht. `\endlinechar` hat normalerweise den Wert 13, was einem <Return> entspricht, wobei ASCII-13 beziehungsweise <Return> normalerweise den Category-Code-5 hat. Danach erst schaut es sich die Zeile genauer an und legt entsprechende Token aufs Fließband.

Was wegen des am Zeilenende eingefügten Category-Code-5-<Return>-Zeichens von dem Männchen gemacht wird, hängt – wie schon erwähnt – vom Status des Leseapparats zu dem Moment ab, wo das Männchen dieses Zeichen wieder betrachtet, und Token aufs Fließband legt.

Steht im Quelltext beispielsweise die Zeile

```
blubb blubb \bla
```

dann ist für die Phrase `\bla` ein Kontrollwort-Token aufs Fließband gelegt worden und der Status des Leseapparats ist auf »Leerzeichen sind zu ignorieren!« gestellt. Das wegen `\endlinechar` hinter `\bla` eingefügte Category-Code-5-<Return>-Zeichen veranlasst das Männchen lediglich, den Rest der Zeile zu ignorieren und sich die nächste Zeile der Eingabe vorzunehmen (und wenn dies denn passiert, den Leseapparatstatus zunächst auf »Neue Zeile einlesen!« umzustellen).

Steht im Quelltext beispielsweise die Zeile

```
blubb blubb
```

dann ist nach dem letzten »b« der Leseapparatstatus auf »Es wird in einer Zeile der Eingabe gelesen!« gestellt. Das wegen `\endlinechar` hinter diesem letzten »b« eingefügte Category-Code-5-<Return>-Zeichen veranlasst das Männchen, ein Space-Token aufs Fließband zu legen, und sich die nächste Zeile der Eingabe vorzuneh-

men. Und wenn dies denn passiert, den Leseapparatstatus zunächst auf »Neue Zeile einlesen!« umzustellen.

Steht im Quelltext beispielsweise die Zeile

```
\macro {arg}
```

dann wird das Männchen für die Phrase `\macro` das entsprechende Kontrollwort-Token aufs Fließband legen und weil gerade ein Kontrollwort-Token aufs Fließband gelegt worden ist, den Leseapparatstatus auf »Leerzeichen sind zu ignorieren!« umstellen. Weil der Leseapparatststus auf »Leerzeichen sind zu ignorieren!« steht, wird das hinter `\macro` stehende Leerzeichen vom Männchen ignoriert.

Das Männchen erblickt dann die geöffnete geschweifte Klammer und legt dafür ein `{`-Character-Token mit Category-Code 1 (begingroup) aufs Fließband und schaltet den Leseapparatstatus auf »Es wird in einer Zeile der Eingabe gelesen!«.

Das Männchen erblickt dann »a« und legt dafür ein a-Character-Token mit Category-Code 11 (Letter) aufs Fließband und schaltet den Leseapparatstatus auf »Es wird in einer Zeile der Eingabe gelesen!«.

Das Männchen erblickt dann »r«und legt dafür ein r-Character-Token mit Category-Code 11 (Letter) aufs Fließband und schaltet den Leseapparatstatus auf »Es wird in einer Zeile der Eingabe gelesen!«.

Das Männchen erblickt dann »g«und legt dafür ein g-Character-Token mit Category-Code 11 (Letter) aufs Fließband und schaltet den Leseapparatstatus auf »Es wird in einer Zeile der Eingabe gelesen!«.

Das Männchen erblickt dann die schließende geschweifte Klammer, und legt dafür ein `}`-Character-Token mit Category-Code 2 (endgroup) aufs Fließband und schaltet den Leseapparatstatus auf »Es wird in einer Zeile der Eingabe gelesen!«.

Dass das wegen `\endlinechar` hinter dieser schließenden geschweiften Klammer eingefügte Category-Code-5-<Return>-Zeichen veranlasst das Männchen, ein Space-Token aufs Fließband zu legen und sich die nächste Zeile der Eingabe vorzunehmen (und wenn dies denn passiert, den Leseapparatstatus zunächst auf »Neue Zeile einlesen« umzustellen).

Steht im Quelltext beispielsweise die Zeile

```
\macro {arg}%
```

dann wird das Männchen für die Phrase `\macro` das entsprechende Kontrollwort-Token aufs Fließband legen und weil gerade ein Kontrollwort-Token aufs Fließband gelegt worden ist, den Leseapparatstatus auf »Leerzeichen sind zu ignorieren!« umstellen.

Weil der Leseapparatststus auf »Leerzeichen sind zu ignorieren!« steht, wird das hinter `\macro` stehende Leerzeichen vom Männchen ignoriert.

Das Männchen erblickt dann die geöffnete geschweifte Klammer, und legt dafür ein {-Character-Token mit Category-Code 1 (begingroup) aufs Fließband und schaltet den Leseapparatstatus auf »Es wird in einer Zeile der Eingabe gelesen!«.

Das Männchen erblickt dann »a« und legt dafür ein a-Character-Token mit Category-Code 11 (Letter) aufs Fließband und schaltet den Leseapparatstatus auf »Es wird in einer Zeile der Eingabe gelesen!«.

Das Männchen erblickt dann »r« und legt dafür ein r-Character-Token mit Category-Code 11 (Letter) aufs Fließband und schaltet den Leseapparatstatus auf »Es wird in einer Zeile der Eingabe gelesen!«.

Das Männchen erblickt dann »g« und legt dafür ein g-Character-Token mit Category-Code 11 (Letter) aufs Fließband und schaltet den Leseapparatstatus auf »Es wird in einer Zeile der Eingabe gelesen!«.

Das Männchen erblickt dann die schließende geschweifte Klammer, und legt dafür ein }-Character-Token mit Category-Code 2 (endgroup) aufs Fließband und schaltet den Leseapparatstatus auf »Es wird in einer Zeile der Eingabe gelesen!«.

Das Männchen erblickt dann das Prozentzeichen, welches den Category-Code 14 (comment) hat. Das Männchen legt kein Token aufs Fließband, sondern macht weiter, indem es sich nicht weiter mit der Zeile befasst. (Wenn sich anschickt, sich mit der nächsten Zeile zu befassen, wird es den Leseapparatstatus zunächst auf »Neue Zeile einlesen!« umzustellen). Das wegen `\endlinechar` hinter dem Prozentzeichen eingefügte Category-Code-5-<Return>-Zeichen wird also ignoriert beziehungsweise dient dem Männchen nicht als Anlass, Space-Token oder `\par`-Token aufs Fließband zu legen.

Wir betrachten folgendes Dokument:

```
\documentclass[12pt,a4paper]{book}
\usepackage[utf8]{inputenc}
\usepackage{graphicx}
\newcounter{strctz}
\DeclareRobustCommand{\strct}[1]{% % %
\refstepcounter{strctz}
\textbf{#1}\textbf{ \Large ({\thestrctz})}\\[2mm]
\rule{2cm}{5mm}
\DeclareRobustCommand{\cstr#1}{\thestrctz}
}
\begin{document}
 \strct{mevalonate}
 (\cstrmevalonate) und
 \strct{17ohpregnenolone}
```

```
 Structure (\cstr17ohpregnenolone) and structure (\cstrmevalonate)
\end{document}
```

Hinter der \refstepcounter-Anweisung wird beim Einlesen und Tokenizen der Eingabe ein Space-Token entstehen und selbiges Space-Token wird in der Ausgabe (dvi/pdf) einen horizontalen Zwischenraum (Leerzeichen) bewirken. Abhilfe schafft ein Comment-Character am Zeilenende:

```
\refstepcounter{strctz}%
\textbf{#1}\textbf{ \Large ({\thestrctz})}\\[2mm]
```

1. Das Leerzeichen nach der geöffneten geschweiften Klammer der zweiten \textbf-Anweisung wird beim Einlesen und Tokenizen der Eingabe als Space-Token getokenized werden und selbiges Space-Token wird in der Ausgabe (dvi/pdf) einen horizontalen Zwischenraum (Leerzeichen) bewirken.
2. Hinter der \\[2mm]-Anweisung wird beim Einlesen und Tokenizen der Eingabe ein Space-Token entstehen. Es wird in der Ausgabe (dvi/pdf) einen horizontalen Zwischenraum (Leerzeichen) bewirken.

Abhilfe schafft ein Comment-Character am Zeilenende:

```
\textbf{#1}\textbf{ \Large ({\thestrctz})}\\[2mm]%
```

oder

```
\textbf{#1}\textbf{\Large ({\thestrctz})}\\[2mm]%
```

oder

```
\textbf{#1}~\textbf{\Large ({\thestrctz})}\\[2mm]%
```

Je nachdem, welche von diesen Zwischenräumen man tatsächlich haben will und ob sie auch durch einen Zeilenumbruch ersetzt werden dürfen oder nicht.

```
\rule{2cm}{5mm}
```

Hinter der \rule-Anweisung wird beim Einlesen und Tokenizen der Eingabe ein Space-Token entstehen. Es wird in der Ausgabe (dvi/pdf) einen horizontalen Zwischenraum (Leerzeichen) bewirken. Abhilfe schafft ein Comment-Character am Zeilenende:

```
\rule{2cm}{5mm}%

\DeclareRobustCommand{\cstr#1}{\thestrctz}
```

Hinter der `\DeclareRobustCommand`-Anweisung wird beim Einlesen und Tokenizen der Eingabe ein Space-Token entstehen. Es wird in der Ausgabe (dvi/pdf) einen horizontalen Zwischenraum (Leerzeichen) bewirken.

Abgesehen davon erwartet `\DeclareRobustCommand` als erstes Argument ein (noch nicht definiertes) Kontrollsequenz-Token (entweder Kontrollsymbol oder Kontrollwort).

Im ersten Argument von `\DeclareRobustCommand` stehen nach dem Einlesen und Tokenizen der Eingabe beziehungsweise nach dem Einlesen und Tokenizen der Definition von `\strct` aber drei Token:

1. Kontrollwort-Token: `\cstr`.
2. Catcode-6-Character-Token (Parameter): `#`
3. Catcode-12-Character-Token (Other): `1`

Beim Expandieren/Ausführen von `\strct` wird `#1` dann durch andere Token ersetzt sein und zwar durch diejenigen, die sich beim Einlesen und Tokenizen des Arguments von `\strct` ergeben.

Zu dem Zeitpunkt, zu dem `\strct` expandiert wird, ist jedenfalls das Einlesen und Tokenizen der Eingabe, also auch der Argumente von `\strctz`, schon längst vorbei. Wenn man also mit Makroargumenten wie »`#1`« jongliert, dann jongliert man zeitmäßig letzten Endes immer bereits mit fertigen Token, nicht mit Dingen, die (ohne weitere Tricks wie `\csname` oder `\string`) Teile des Namens eines (noch zu erzeugenden) Kontrollsequenz-Token sein könnten.

Also zu den »weiteren Tricks«:

Wenn es darum geht, anhand des Makroargumentes und der vorangehenden `cstr`-Phrase dem `\DeclareRobustCommand`-Makro ein Kontrollsequenz-Token als Argument zu liefern, dann muss man dieses Kontrollsequenz-Token zuvor anhand bereits vorliegender Token »basteln« lassen – beispielsweise mittels `\expandafter` und `\csname`…`\endcsname`:

```
\expandafter\DeclareRobustCommand\expandafter{%
  \csname cstr#1\endcsname}{\thestrctz}
```

Wenn es zusätzlich noch darum geht, nicht nur das Kontrollsequenz-Token `\thestrctz` in den Definitionstext zu kriegen, sondern die komplette Expansion davon, empfiehlt es sich, dieses Token vorher (beispielsweise mittels `\protected@edef`) komplett in den Definitionstext einer anderen Kontrollsequenz hineinzuexpandieren, die dann ihrerseits hoffentlich das Gewünschte nach genau einem Expansionsschritt beziehungsweise nach genau einer `\expandafter`-Kette liefert.

Ein weiteres Problem sehe ich noch. Um es zu erklären, möchte ich wieder weiter ausholen:

Wenn TEX Programmzeilen (entweder aus Datei oder von Konsole) einliest, dann erzeugt es Kontrollsequenz-Token auf zwei Arten:

1. Kontrollsymbol-Token werden erzeugt wenn in der Eingabe ein Character mit Category-Code 0 (Escape), normalerweise der Backslash, gefunden wird, direkt gefolgt von einem Character, der nicht den Category-Code 11 hat. Der Name des Kontrollsymbol-Tokens entspricht diesem einen nicht-Category-Code-11-Character. Was in der Eingabe darauf folgt, wird nicht mehr dem Namen des zu erzeugenden Kontrollsymbol-Tokens zugerechnet, sondern anderweitig getokenized oder (beispielsweise im Fall von Comment-Character) anderweitig verarbeitet ohne die Entstehung von Token zu bewirken, jedoch das Verhalten des Lese- und Tokenize-Apparates beeinflussen.
2. Kontrollwort-Token werden erzeugt, wenn in der Eingabe ein Character mit Category-Code 0 (escape), normalerweise der Backslash, gefunden wird, direkt gefolgt von einer Sequenz von Characters, die alle den Category-Code 11 (Letter) haben. Der Name des Kontrollwort-Tokens entspricht dieser Category-Code-11-Character-Sequenz. Was in der Eingabe darauf folgt, wird nicht mehr dem Namen des zu erzeugenden Kontrollwort-Tokens zugerechnet, sondern anderweitig getokenized oder (beispielsweise im Fall von Comment-Character) anderweitig verarbeitet ohne die Entstehung von Token zu bewirken, jedoch eventuell das Verhalten des Lese- und Tokenize-Apparates beeinflussen.

Ziffern wie 1 und 7 haben Category-Code 12 (Other).

Wenn TEX also bei Einlesen und Tokenizen der Eingabe einen Backslash und eine darauffolgende Sequenz aus Category-Code-11-Characters, wie beispielsweise »cstr« und daraufhin einen Category-Code-12-Character »1« findet, wird es die Sache tokenizen als ein Kontrollwort `\cstr` und ein Category-Code-12-Character-Token »1«.

Man kann also Kontrollwörter mit einer Mixtur von Zeichen im Namen, die zum Zeitpunkt des Einlesens und Tokenizens des entsprechenden Quelltexts nicht alle den Category-Code 11 (Letter) haben, nicht durch Einlesen und Tokenizen der Eingabe erzeugen lassen.

Eine Sequenz im Quelltext wie `\cstr17ohpregnenolone` wird nicht aufgefasst/getokenized werden als ein Kontrollwort-Token mit dem Namen »cstr17ohpregnenolone« sondern als ein Kontrollwort-Token mit dem Namen »cstr«, gefolgt von diversen Character-Token.

Aber man kann solche Kontrollwort-Token nach dem Einlesen und Tokenizen mittels `\csname…\endcsname` erzeugen lassen.

`\csname` ist ein Kontrollwort-Token. Es ist ein expandibles Primitive, welches nach dem Einlesen der Eingabe beziehungsweise beim Tokenizen der Eingabe entsteht

und danach irgendwann ausgeführt/expandiert wird und dann seinerseits eine Sequenz von Token als Namen einer Kontrollsequenz auffasst. Die Expansion beziehungsweise der Ersetzungstext dieses \csname-Primitive besteht dann aus dem entsprechenden Kontrollsequenz-Token. TeX geruht übrigens, während es den Namen für \csname »zusammensucht«, zu dem es das entsprechende Kontrollsequenz-Token liefern soll, expandible Token zu expandieren.

So ergibt beispielsweise das \csname-Konstrukt in

```
\def\macroA{\macroB\macroC\endcsname}
\def\macroB{Te}
\def\macroC{X}
\csname \macroA
\bye
```

keinen »missing \endcsname«-Fehler sondern das Kontrollwort-Token »TEX«.

Auch lustig ist folgendes:

```
\def\macroA{\macroB\macroC endcsname\endcsname}
\def\macroB{Te}
\def\macroC{X\csname}
\csname \macroA
\bye
```

Man kann also einen Mechanismus basteln, bei dem man nicht das Token mit Namen »cstr17ohpregnenolone« »direkt eingibt«, sondern ein Makro \cstr aufruft, welches ein Argument nimmt und aus diesem mittels \csname…\endcsname das entsprechende Token bastelt:

Insgesamt also:

```
\documentclass[12pt,a4paper]{book}
\makeatletter
%\usepackage[utf8]{inputenc}
\usepackage{graphicx}
\newcounter{strctz}
\DeclareRobustCommand{\strct}[1]{%
  \refstepcounter{strctz}%
  \textbf{#1}\textbf{\Large({\thestrctz})}%
  \\[2mm]%
  \rule{2cm}{5mm}%
  \expandafter\DeclareRobustCommand
  \csname crf#1\expandafter\endcsname
  \expandafter{\number\value{strctz}}%
  % globalisieren:
```

```
  \global\expandafter\let\csname crf#1\expandafter\endcsname
    \csname crf#1\endcsname
}

\DeclareRobustCommand{\crf}[1]{%
  \expandafter\ifx\csname crf#1\endcsname\relax
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
  {%
    \protect\G@refundefinedtrue
    \nfss@text{\reset@font\bfseries ??}%
    \@latex@warning {Reference `#1' on page \thepage \space undefined}%
  }{%
    \csname crf#1\endcsname
  }%
}
\makeatother
\begin{document}
 \strct{mevalonate}
 (\crf{mevalonate}) und

 \strct{17ohpregnenolone}

 Structure (\crf{17ohpregnenolone}) and structure (\crf{mevalonate}) and
Structure (\crf{17ohpregnenolone})

 \strct{alanine}  and (\crf{alanine}) (\crf{mevalonate}) (\crf{17ohpregnenolone})
\end{document}
```

Der Umstand, dass das Männchen zeilenweise liest, impliziert übrigens, dass ein in eine Zeile geschriebenes Assignment, um `\endlinechar` und damit das Zeichen, das am Ende von Zeilen angefügt wird, zu ändern, sich nicht bereits auf diese Zeile auswirkt, sondern erst auf die folgende Zeile.

Das liegt daran, dass diejenige Zeile, die das Assignment enthält, bereits eingelesen (und deshalb auch bereits mit einem `\endlinechar` versehen) und getokenized sein muss, bevor TEX daran gehen kann, die anhand dieser Zeile erzeugten Token zu verarbeiten und dabei dann beispielsweise Assignments auszuführen.

Im folgenden Beispiel wird mittels der Notation nach dem Schema[2] (die man bei TEX in Zusammenhang mit dem Einlesen von Zahlenwerten auch nehmen kann):

[2] Im TEXbook ist dabei von »alphabetic constant« die Rede.

`\`⟨einzelnes Zeichen als Kontrollsequenz⟩`

TEX zunächst die Anweisung gegeben, dem `\endlinechar`-Parameter den Wert/Character-Code des Zeichens »X« zu geben. In der nächsten Zeile, die der Zeile mit dieser Anweisung folgt, wird deshalb die am Schluss stehende Zeichensequenz »`\Te`« zu »`\TeX`« erweitert und dafür im weiteren Verlauf dann ein Kontrollwort-Token mit Namen »TeX« erzeugt. In dieser nächsten Zeile steht am Anfang allerdings auch die Anweisung, dem `\endlinechar`-Parameter den Wert/Character-Code des Zeichens »p« zu geben. Deshalb wird in der übernächsten Zeile die am Schluss stehende Zeichensequenz »`\endgrou` « zu »`\endgroup`« erweitert und dafür dann im weiteren Verlauf ein Kontrollwort-Token mit Namen »endgroup« erzeugt.

Dann habe ich mir noch den Spaß erlaubt, die Phrase »Hello 100 wild 13 world!« mittels `\endlinechar`-Spielerei erzeugen zu lassen.

Hier sieht man in der Ausgabedatei, dass Dank `\endlinechar` beim Einlesen am Ende der Zeilen das Zeichen »d« angefügt worden ist und erst *danach* anhand dieser Zeilen Token erzeugt und verarbeitet worden sind. Insbesondere ist die folgende Zeile zu beachten:

```
{} \endgroup\the\Macro{endlinechar}{} worl
```

Dank `\endgroup` ist hier beim Verarbeiten der aus der Sequenz `\the\endlinechar` entstehenden Token der Wert des `\endlinechar` vom Wert `\d ( = 100 ) wieder zurückgesetzt auf den Wert 13, sodass `\the\endlinechar` die Sequenz »13« liefert. Aber trotzdem steht statt der Phrase »worl « die Phrase »world« da:

Zum Zeitpunkt des Einlesens dieser Zeile war `\endlinechar` eben noch auf den Wert `\d (= 100) gesetzt, also wurde am Ende der Zeile ein Zeichen »d« angefügt.

Als danach dann die anhand dieser Zeile entstandene Tokensequenz verarbeitet wurde, wurde das `\endgroup`-Kontrollwort-Token verarbeitet und dadurch der `\endlinechar`-Parameter wieder auf seinen alten Wert 13 zurückgesetzt. Danach wurden dann die Kontrollwort-Token `\the` und `\endlinechar` verarbeitet und lieferten für die Ausgabedatei als Expansion die Sequenz »13« .

Beispiel:

```
%Der \Macro{endlinechar}-Integer-Parameter hat
%normalerweise den Wert 13.

\begingroup \Macro{endlinechar}=`\X\relax
\Macro{endlinechar}=`\p \Te
\endgrou

hello \begingroup\Macro{endlinechar}=`\d\relax
```

```
\the\Macro{endlinechar}{} wil
{} \endgroup\the\Macro{endlinechar}{} worl
!

\bye
```

Man kann auch Leute verwirren, indem man `\endlinechar` missbraucht, um Catcode-1-(begingroup)- oder Catcode-2-(endgroup)-Character-Token erzeugen zu lassen:

Beispiele

```
\def\Schreibmaschine#1{{\tt|#1|}}%
{\Macro{endlinechar}=`\{\relax
}\Schreibmaschine
Anfangsklammer des Arguments von \string\Schreibmaschine{} wird
dank \string\Macro{endlinechar}{} erzeugt.}
\par Das hier gehoert nicht mehr zum Argument.
\bye
```

```
\def\Schreibmaschine#1{{\tt|#1|}}%
{\Macro{endlinechar}=`\}\relax
}\Schreibmaschine{Argument von \string\Schreibmaschine{} geht bis Zeilenende.
\par Das hier gehoert nicht mehr zum Argument.
\bye
```

Wer zur Strafe etwas zehn mal schreiben soll, kann das mit TeX auch machen, indem er zunächst einem Zeichen den Catcode 13 (active) zuweist.

Active-Character-Token haben die Eigenschaft, dass man damit weitgehend umgehen kann wie mit Kontrollsequenz-Token. Man kann sie in Assignments verwenden und damit beispielsweise Makros definieren. Dann sind sie expandibel.

Dann definiert dieser Jemand, dass das aktive Zeichen zur zu schreibenden Phrase expandieren soll.

Dann macht er dieses Zeichen zum `\endlinechar` und gibt entsprechend Leerzeilen vor.

Beispiel:

```
\begingroup
\catcode`\Y=13\relax
\def Y{Ich muss das zehnmal schreiben.\par}
\Macro{endlinechar}=`\Y
% Nach den Kommentaren folgen neun Leerzeilen, die dank \endlinechar
% jeweils ein Y an ihrem Ende angefügt bekommen.
% Die Zeile mit \expandafter\endgroup bekommt auch ein Y an ihrem
```

```
% Ende angefügt.
% Das \expandafter wird gebraucht, damit dieses Y mit Catcode 13(active)
% getokenized und expandiert wird, und nicht mit Catcode 11(letter).
% Ausserdem ist die Definition von Y nur so lange gültig, wie der durch
% \begingroup erzeugte "local scope" besteht und das \expandafter sorgt
% dafür, dass Y expandiert wird bevor dieser local scope durch \endgroup
% terminiert wird.

\expandafter\endgroup
Fertig.
\bye
```

Kursive Zeichen mit Überstrich[3]

The slanting makes the correct length of the bar a little more complicate. The following example measures the width of an upright X and uses this for the length of the bar. The solution also works for different math styles:

Default: $\bar{X}$ $\overline{X}$ New: $\overline{X}\overline{X}\overline{X}$

```
[...]
\usepackage{amsmath,amsfonts,mathtools}
\makeatletter
\newcommand*\Xbar{}%
\DeclareRobustCommand*\Xbar{\mathpalette\@Xbar{}}
\newcommand*{\@Xbar}[2]{%
  % #1: math style #2: unused (empty)
  \sbox0{$#1\mathrm{X}\m@th$}%
  \sbox2{$#1X\m@th$}%
  \rlap{%
    \hbox to\wd2{\hfill$\overline{\vrule width 0pt height\ht0 \kern\wd0}$}%
  }%
  \copy2
}
\makeatother
[...]

Default: {\huge$\bar{X}$ $\overline{X}$}\quad
New: {\huge$\Xbar\scriptstyle\Xbar\scriptscriptstyle\Xbar$}
```

[3] Heiko Oberdiek am 11. 6. 2016 in http://tex.stackexchange.com/questions/314238/bar-and-overline

Neue Pakete auf CTAN

Jürgen Fenn

Der Beitrag stellt neue Pakete auf CTAN seit der letzten Ausgabe.Bloße Updates können auf der moderierten *CTAN-ann*-Mailingliste verfolgt werden, die auch auf Twitter als `@ctanannounce` verfügbar ist.

binarytree von *Aleksandrina Nikolova* stellt Befehle bereit, um Binärbäume mit Ti*k*Z zu zeichnen.
`CTAN:graphics/pgf/contrib/binarytree`

randomlist von *Jean-Côme Charpentier* ist ein Paket, mit dem man Arbeitsblätter und Klausuren erstellen kann, die individuell auf den jeweiligen Schüler zugeschnitten sind.
`CTAN:macros/generic/randomlist`

mgltex von *Diego Sejas Viscarra* und *Alexey Balakin* erlaubt es, Skripte in der Sprache MGL in LaTeX-Dokumente einzubetten. So kann man Abbildungen mit der Bibliothek MathGL automatisch erzeugen und in Dokumente übernehmen.
`CTAN:graphics/mgltex`

texosquery von *Nicola Talbot* ist ein Java-Programm, mit dem man per *shell escape* Informationen über das jeweilige Betriebssystem abfragen kann.
`CTAN:support/texosquery`

nodetree von *Josef Friedrich* ist ein Paket für LuaTeX-Entwickler, das Lua-Node-Trees darstellen kann.
`CTAN:macros/luatex/generic/nodetree`

navydocs von *Peter A. Rochford* dient dazu, das Layout von technischen Berichten der amerikanischen Marine bereitzustellen.
`CTAN:macros/latex/contrib/navydocs`

biblatex-ijsra von *Lukas C. Bossert* ist ein BibLaTeX-Stil für das *International Journal of Student Research in Archaeology*.
`CTAN:macros/latex/contrib/biblatex-contrib/biblatex-ijsra`

milog von *Josef Kleber* hilft beim Erstellen der Dokumentation der Arbeitszeiten von Arbeitnehmern, wie sie nach dem deutschen Mindestlohngesetz vorgeschrieben ist.
`CTAN:macros/latex/contrib/milog`

graphics-def vom *LaTeX-Team* fasst die Farb- und Grafik-Options-Dateien aller TeX-Engines nunmehr in einem Paket zusammen, um gemeinsame Updates in Zukunft einfacher zu machen.
`CTAN:macros/latex/contrib/graphics-def`

ecgdraw von *Marco Scavino* und *Ezio Aimé* kann EKG-Wellen zeichnen. Das Paket baut auf Ti*k*Z und LaTeX3 auf.
`CTAN:graphics/ecgdraw`

biblatex-nottsclassic von *Lukas C. Bossert* setzt die Zitierregeln der *University of Nottingham* um.
CTAN:macros/latex/contrib/biblatex-contrib/biblatex-nottsclassic

OldStandardT1 von *Bob Tennent* enthält die Schriftart Old Standard von *Alexey Kryukov* samt der dazugehörigen LaTeX-Unterstützung.
CTAN:fonts/oldstandardt1

fvextra von *Geoffrey Poore* stellt eine Reihe von Erweiterungen und Verbesserungen zu dem Paket fancyvrb bereit, insbesondere einen automatischen Zeilenumbruch und einen verbesserten Mathematiksatz.
CTAN:macros/latex/contrib/fvextra

gitfile-info von *André Hilbig* ermöglicht den Zugriff auf die Metadaten einzelner Dateien in einem Git-Repositorium.
CTAN:support/gitfile-info

diffcoeff von *Andrew Parsloe* enthält die Stile `diffcoeff.sty` und `diffcoeffx.sty`, um Differenzialkoeffizienten zu setzen.
CTAN:macros/latex/contrib/diffcoeff

olsak-misc von *Petr Olšák* ist eine Sammlung verschiedener Plain-TeX-Makros.
CTAN:macros/generic/olsak-misc

aucklandthesis von *Alistair Kwan* ist eine Klasse für Master- und Doktorarbeiten an der australischen Universität Auckland.
CTAN:macros/latex/contrib/aucklandthesis

autobreak von *Takahiro Ueda* kann lange mathematische Formeln, die mit der align-Umgebung aus dem amsmath-Paket gesetzt werden, zeilen- und seitenweise umbrechen.
CTAN:macros/latex/contrib/autobreak

hustthesis von *Xu Cheng* ist eine inoffizielle Vorlage für Abschlussarbeiten an der chinesischen Universität Huazhong.
CTAN:macros/latex/contrib/hustthesis

pstricks-examples-7 von *Herbert Voß* enthält die Beispiele aus der siebten Auflage seines Buchs *PSTricks – Grafik mit PostScript für TeX und LaTeX*.
CTAN:info/examples/PSTricks_7_de

umbclegislation von *Lin DasSarma* ist eine Klasse für *legislation files* der *UMBC Student Government Association*.
CTAN:macros/latex/contrib/umbclegislation

ietfbibs von *Richard Mortier* enthält die überarbeitete Fassung des Awk-Skripts `rfc2bib`, um BibTeX-Dateien aus IETF-Index-Files zu erzeugen.
CTAN:biblio/bibtex/utils/misc/ietfbibs

markdown von *Vít Novotný* (mit Vorarbeiten von *John MacFarlane* und *Hans Hagen*) kann Markdown-Texte zur Verwendung in Plain TeX, LaTeX und ConTeXt konvertieren und rendern.
CTAN:macros/generic/markdown

axodraw2 von *John Collins* und *J. A. M. Vermaseren* ist eine Variante des Pakets axodraw zum Zeichnen von Feynman-Diagrammen. Im Vergleich zu seinem Vorgänger bietet es flexiblere grafische Möglichkeiten und kann mit LaTeX/dvips sowie mit pdflatex verwendet werden. Für XeLaTeX und LuaTeX gibt es das Hilfsprogramm axohelp für die geometrischen Berechnungen.
CTAN:graphics/axodraw2

sanitize-umlaut von *Thomas F. Sturm* ermöglicht es, bei Verwendung von pdfLaTeX und makeindex Umlaute und das »scharfe S« in \index-Befehlen direkt einzugeben. Hierzu werden die entsprechenden Kodierungen und Makros in inputenc und babel geändert.
CTAN:macros/latex/contrib/sanitize-umlaut

xcntperchap von *Christian Hupfer* ist eine Neufassung des Pakets cntperchap desselben Autors. Es gibt die Zählerwerte für Kapitel, Abschnitte, Gleichungen usw. für das ganze Dokument aus.
CTAN:macros/latex/contrib/xcntperchap

cquthesis von *Li Zhennan* ist eine Klasse für Abschlussarbeiten an der chinesischen Universität Chongqing.
CTAN:macros/latex/contrib/cquthesis

getargs von *Steven B. Segletes* ist ein Listenparser, der Verbesserungen im Vergleich zu den Paketen stringstrings und readarray desselben Autors bietet.
CTAN:macros/latex/contrib/getargs

churchslavonic von *Aleksandr Andreev* und *Mike Kroutikov* stellt die LaTeX-Unterstützung für Texte bereit, die mit den Schriften aus fonts-churchslavonic in Kirchenslawisch (mit LuaTeX oder XeTeX) gesetzt werden.
CTAN:language/churchslavonic

fonts-churchslavonic von *Aleksandr Andreev*, *Yuri Shardt* und *Nikita Simmons* enthält vier Schriften in den Formaten TrueType und OpenType zum Setzen von Texten in Kirchenslawisch, der Liturgiesprache der orthodoxen Kirchen und der katholischen Ostkirchen.
CTAN:fonts/fonts-churchslavonic

biblatex-iso690 von *Michal Hoftich* ist ein BibLaTeX-Stil nach den Vorgaben der Norm ISO 690 in der Fassung aus dem Jahr 2010, wie sie an tschechischen Universitäten verwendet wird.
CTAN:macros/latex/contrib/biblatex-contrib/biblatex-iso690

biblatex-abnt von *Daniel B. Marques* ist ein BibLaTeX-Stil nach den Vorgaben der brasilianischen Institution für technische Normung ABNT.
CTAN:macros/latex/contrib/biblatex-contrib/biblatex-abnt

coloring von *Zou Hu* ist hilfreich beim Definieren eigener Farbnamen für tikz-Grafiken und beamer-Themes.
CTAN:macros/latex/contrib/coloring

dvisvgm-def von *Till Tantau* enthält die Unterstützung zu dem Treiber dvisvgm für die Pakete graphics und color.
CTAN:macros/latex/contrib/dvisvgm-def

makebase von *Peter Flynn* stellt LaTeX-Zähler auf einer anderen Basis als zehn dar. Standardmäßig wird die Basis 16 gewählt, andere Werte sind möglich; getestet wurde das Paket bisher nur für den Zähler page.
CTAN:macros/latex/contrib/makebase

acmart von *Boris Veytsman* ist die neue Klasse zum Setzen von Beiträgen in den Veröffentlichungen der *Association for Computing Machinery (ACM)*. Sie ersetzt alle früher von der Vereinigung verwendeten LaTeX-Klassen.
CTAN:macros/latex/contrib/acmart

datepicker-pro von *D. P. Story* stellt ein Makro bereit, mit dem man ein Formularfeld zur Eingabe des Datums in amerikanischer Notation in eine PDF-Datei einfügen kann. Setzt Adobe Acrobat Reader voraus.
CTAN:macros/latex/contrib/datepicker-pro

chivo von *Arash Esbati* enthält die Fonts und die LaTeX-Unterstützung für die freie serifenlose Schriftart Chivo von *Héctor Gatti* und der Firma *Omnibus Type*.
CTAN:fonts/chivo

ptex-base von *Kazuki Maeda* ist das Plain-TeX-Format und die Dokumentation zu pTeX und e-pTeX.
CTAN:language/japanese/ptex-base

ptex-fonts von *Kazuki Maeda* liefert Font-Metriken, die man mit pTeX verwenden kann.
CTAN:fonts/ptex-fonts

platex-base von *Kazuki Maeda* enthält pLaTeX 2_ϵ für pTeX und weitere Makros.
CTAN:language/japanese/platex-base

uplatex-base von *Kazuki Maeda* stellt pLaTeX 2_ϵ und diverse weitere Makros für upTeX bereit.
CTAN:language/japanese/uplatex-base

uptex-base von *Kazuki Maeda* enthält das Plain-TeX-Format für upTeX.
CTAN:language/japanese/uptex-base

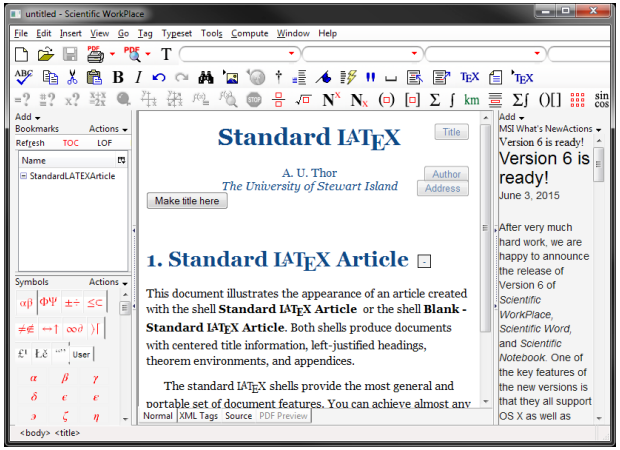
pbibtex-base von *Kazuki Maeda* bietet mehrere Bibliografiestile zur Verwendung mit pBibTeX für Texte in japanischer Sprache.
CTAN:biblio/pbibtex/base

uptex-fonts von *Kazuki Maeda* enthält die Font-Metriken zur Verwendung von Schriften mit upTeX.
CTAN:fonts/uptex-fonts

rosario von *Arash Esbati* enthält die die LaTeX-Unterstützung für die freie Schriftart Rosario von *Héctor Gatti* und der Firma *Omnibus Type*.
CTAN:fonts/rosario

Technisch-wissenschaftliches Publizieren

Die MacKichan-Produktfamilie Scientific WorkPlace, Scientific Word und Scientific Notebook bietet integrierte Arbeitsumgebungen für das Schreiben und Publizieren von technisch-wissenschaftlichen Dokumenten und - in im Fall von Scientific WorkPlace und Scientific Notebook - das interaktive Berechnen innerhalb von Dokumenten.



Neu in Version 6:

- Kompatibilität zu Mac OS X
- Neues Erscheinungsbild
- Einfaches Teilen von Dokumenten
- Erleichterte Dokumenterstellung:
  - Rückgängig machen von beliebig vielen Arbeitsschritten
  - Schnelle Formatierung von Dokumenten
  - u.v.m.

MacKichan SOFTWARE, INC.

Nähere Informationen erhalten Sie unter:
workplace@additive-net.de oder
www.additive-workplace.de

ADDITIVE
SOFT- & HARDWARE FÜR TECHNIK & WISSENSCHAFT

Spielplan

2016

20. 8. – 21. 8. **FrOSCon**, Free and Open Source Conference
Hochschule Bonn-Rhein-Sieg
Grantham-Allee 20, 53757 Sankt Augustin
https://www.froscon.de/startseite/

10. 9. **Herbsttagung**
und 55. Mitgliederversammlung von DANTE e.V.
Georg-August-Universität Göttingen
http://www.dante.de/events/Herbst2016.html

25. 9. – 1. 10. **10th International ConTEXt Meeting**
Kalenberg (Niederlande)
http://meeting.contextgarden.net/2016/

5. 11. – 6. 11. **OpenRheinRuhr**
Freie Software und Netzpolitik
Rheinisches Industriemuseum (RIM) Oberhausen
http://www.openrheinruhr.de/

2017

22. 3. – 24. 3. **DANTE 2017**
und 56. Mitgliederversammlung von DANTE e.V.
Deutsches Elektronen-Synchotron (DESY)
Platanenallee 6, 15738 Zeuthen

Stammtische

In verschiedenen Städten im Einzugsbereich von DANTE e.V. finden regelmäßig Treffen von TEX-Anwendern statt, die für jeden offen sind. Im WWW gibt es aktuelle Informationen unter http://projekte.dante.de/Stammtische/WebHome.

Aachen
Torsten Bronger,
bronger@physik.rwth-aachen.de
Gaststätte Knossos, Templergraben 28, 52062 Aachen
Zweiter Donnerstag im Monat, 19.00 Uhr

Berlin
Michael-E. Voges, Tel.: (03362) 50 18 35,
mevoges@t-online.de
Mantee – Café Restaurant, Chausseestraße 131, 10115 Berlin
Zweiter Donnerstag im Monat, 19.00 Uhr

Bremen
Winfried Neugebauer, Tel.: 0176 60 85 43 05,
tex@wphn.de
Wechselnder Ort
Erster Donnerstag im Monat, 18.30 Uhr

Dresden
Daniel Borchmann, daniel@algebra20.de, http://tug-dd.kxpq.de/Home
auf Anfrage

Erlangen
Walter Schmidt, Peter Seitz,
w.a.schmidt@gmx.net
Gaststätte »Deutsches Haus«, Luitpoldstraße 25, 91052 Erlangen
Dritter Dienstag im Monat, 19.00 Uhr

Frankfurt
Harald Vajkonny,
http://wiki.lug-frankfurt.de/TeXStammtisch
Restaurant »Zum Jordan«, Westerbachstr. 7, 60489 Frankfurt
Zweimonatlich, Vierter Donnerstag im Monat, 19.30 Uhr

Göttingen
Holger Nobach,
holger.nobach@nambis.de, http://goetex.nambis.de/
Restaurant Mazzoni Cucina Italiana,
Hermann-Rein-Straße 2, 37075 Göttingen
Dritter Donnerstag im Monat, 18.00 Uhr

Hamburg
Lothar Fröhling,
lothar@thefroehlings.de
Letzter Dienstag im Monat an wechselnden Orten, 19.00 Uhr

Hannover
Mark Heisterkamp,
heisterkamp@rrzn.uni-hannover.de
Seminarraum RRZN, Schloßwender Straße 5, 30159 Hannover
Zweiter Donnerstag im Monat, 18.30 Uhr

Heidelberg
Martin Wilhelm Leidig, Tel.: 0170 418 33 29,
moss@moss.in-berlin.de
Anmeldeseite zur Mailingliste: http://tinyurl.com/stammtisch-HD
Wechselnder Ort
Letzter Freitag im Monat, ab 19.30 Uhr

Karlsruhe
Klaus Braune, Tel.: (0721) 608-4 40 31,
klaus.braune@kit.edu,
SCC (Steinbuch Centre for Computing) des KIT (vormals Universität Karlsruhe, Rechenzentrum),
Zirkel 2, 2. OG, Raum 203, 76131 Karlsruhe
Erster Donnerstag im Monat, 19.30 Uhr

Köln
Uwe Ziegenhagen
Dingfabrik, Erzbergerplatz 9, 50733 Köln
Zweiter Dienstag im Monat, 19.00 Uhr

München
Uwe Siart,
uwe.siart@tum.de, http://www.siart.de/typografie/stammtisch.xhtml
Erste Woche in geradzahligen Monaten an wechselnden Tagen, 19.00 Uhr

Stuttgart
Bernd Raichle,
bernd.raichle@gmx.de
»Trollinger-Stubn«, Rotebühlstr. 50, 70178 Stuttgart
Zweiter Dienstag im Monat, 19.30 Uhr

Trier
Martin Sievers,
ttt@schoenerpublizieren.de
Anmeldeseite zur Mailingliste: http://lists.schoenerpublizieren.de/cgi-bin/mailman/listinfo/ttt
Universität Trier
nach Vereinbarung

Wuppertal
Andreas Schrell, Tel.: (02193) 53 10 93,
as@schrell.de
Restaurant Croatia »Haus Johannisberg«, Südstraße 10, 42103 Wuppertal
Zweiter Donnerstag im Monat, 19.30 Uhr

Würzburg
Bastian Hepp,
LaTeX@sning.de
nach Vereinbarung

Adressen

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

Tel.: (0 62 21) 2 97 66 (Mo., Mi.–Fr., 10.00–12.00 Uhr)
Fax: (0 62 21) 16 79 06
E-Mail: dante@dante.de

Konto: VR Bank Rhein-Neckar eG
IBAN DE67 6709 0000 0002 3100 07
SWIFT-BIC GENODE61MA2

Vorstand

Vorsitzender:	Martin Sievers	president@dante.de
stv. Vorsitzender:	Herbert Voß	vice-president@dante.de
Schatzmeisterin:	Doris Behrendt	treasurer@dante.de
Schriftführer:	Manfred Lotz	secretary@dante.de
Beisitzer:	Harald König	
	Volker RW Schaa	
	Dominik Wagenführ	
	Uwe Ziegenhagen	

Ehrenmitglieder

Peter Sandner	22.03.1990	Klaus Thull († 2012)	22.03.1990
Yannis Haralambous	05.09.1991	Barbara Beeton	27.02.1997
Luzia Dietsche	27.02.1997	Donald E. Knuth	27.02.1997
Eberhard Mattes	27.02.1997	Hermann Zapf († 2015)	19.02.1999
Joachim Lammarsch	12.04.2014	Rainer Schöpf	12.04.2014

Webserver und Mailingliste

DANTE:	http://www.dante.de/	(Rainer Schöpf, Joachim Schrod)
CTAN:	http://mirror.ctan.org/	
DANTE-EV:	https://lists.dante.de/mailman/listinfo/dante-ev	

FAQ

DTK: http://projekte.dante.de/DTK/WebHome
TEX: http://projekte.dante.de/DanteFAQ/WebHome

Autoren/Organisatoren

Lukas C. Bossert [7]
Cranachstr. 24
12157 Berlin
lukas@digitales-altertum.de

Ulrich Diez [29]
eu_angelion@web.de

Jürgen Fenn [41]
Friedensallee 174/20
63263 Neu-Isenburg
juergen.fenn@gmx.de

Heiko Oberdiek [41]
Heiko.Oberdiek@gmail.com

Martin Sievers [4,5]
siehe Seite 50

Herbert Voß [3,7,24,29]
Wasgenstraße 21
14129 Berlin
herbert@dante.de

Uwe Ziegenhagen [7,21]
Lokomotivstr. 9
50733 Köln
ziegenhagen@gmail.com

Die TEXnische Komödie

28. Jahrgang Heft 3/2016 August 2016