Die TEXnische Komödie

dante
Deutschsprachige
Anwendervereinigung TEX e.V.

23. Jahrgang Heft 4/2011 November 2011

4/2011

Impressum

»Die TEXnische Komödie« ist die Mitgliedszeitschrift von DANTE e.V. Der Bezugspreis ist im Mitgliedsbeitrag enthalten. Namentlich gekennzeichnete Beiträge geben die Meinung der Autoren wieder. Reproduktion oder Nutzung der erschienenen Beiträge durch konventionelle, elektronische oder beliebige andere Verfahren ist nicht gestattet. Alle Rechte zur weiteren Verwendung außerhalb von DANTE e.V. liegen bei den jeweiligen Autoren.

Beiträge sollten in Standard-LATEX-Quellcode unter Verwendung der Dokumentenklasse `dtk` erstellt und per E-Mail oder Datenträger (CD) 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 beta-0.70.1-2011061410 (rev 4277) (TEXLive 2011)` erstellt. Als Standard-Schriften kamen TEX Gyre Pagella, TEX Gyre Heros, Bera Mono und Latin Modern Math zum Einsatz.

Erscheinungsweise: vierteljährlich
Erscheinungsort: Heidelberg
Auflage: 2700
Herausgeber: DANTE, Deutschsprachige Anwendervereinigung TEX e.V.
Postfach 10 18 40
69008 Heidelberg

E-Mail: `dante@dante.de`
`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, Rolf Niepraschk, Heiko Oberdiek, Christine Römer, Gert Seidl, Martin Sievers

Redaktionsschluss für Heft 1/2012: 15. Januar 2012 ISSN 1434-5897

Editorial

Liebe Leserinnen und Leser,

Sie halten wieder eine mit LuaLaTeX gesetzte Ausgabe in der Hand, die ebenso wie die letzte wieder eine kleine Herausforderung war, denn der Artikel von Arno Trautmann zu Nodes in LuaTeX lief problemlos mit der Dokumentenklasse `dtk`, jedoch nicht im Zusammenhang mit der Komödie, obwohl auch hier dieselbe Dokumentenklasse verwendet wird. Nach einiger Zeit, die ich mit der Problemlösung verbrachte, war das »schuldige« Paket gefunden, `babel` mit seinen aktiven Zeichen. Das Problem konnte behoben werden, aber aus zeitlichen Gründen verzichtete ich auf die Analyse. Die Anwendung von LuaTeX im »Kleinen« ist sicherlich kein Problem, doch für die Anwendung im »Großen« kann man immer noch mit einigen Überraschungen rechnen.

Ähnliches gibt es auch immer wieder von LaTeX3 zu berichten, welches den Zustand einer »Großbaustelle« noch nicht verlassen hat. Immerhin ist das Entwicklerteam zwischenzeitlich verstärkt worden, sodass endlich regelmäßige Lebenszeichen zu vernehmen sind, indem die `l3`-Pakete mittlerweile in kürzeren zeitlichen Abständen erneuert werden und von Anwendern mit einem TeXLive- oder MiKTeX-Update auf den aktuellen Stand gebracht werden können. Verschiedene Pakete benutzen bereits Makros aus diesen experimentellen Paketen, beispielsweise `fontspec`. Roland Geiger hat den von Joseph Wright verfassten »Fahrplan« zur Entwicklung von LaTeX3 übersetzt. Auf der TUG 2011 berichtete Frank Mittelbach über die Problematik, wobei man den Vortrag in einiger Zeit auf `http://river-valley.tv` sehen und hören kann.

Die Lua-Reihe wird durch die Übersetzung eines Artikels von Manuel Pégourié-Gonnard fortgesetzt, welcher zuerst in der »Cahier GUTenberg« erschien. Der Artikel behandelt den internen Umgang mit Farben unter LuaTeX.

Petra Rübe-Pugliese hat den Artikel aus der letzten Ausgabe zu den experimentellen Trennmustern zum Anlass genommen, Debian-Nutzern Hilfen für deren Installation zu geben, während Philipp Lehman noch einmal eingehender die Problematik im Zusammenhang mit BibTeX und Sonderzeichen beschreibt.

Die Teilnehmer der Herbsttagung in Garmisch-Partenkirchen mussten geahnt haben, dass das Wetter tagsüber aus der Kategorie »Sommer« sein würde, denn mit mehr als 40 Teilnehmern war diese Mitgliederversammlung außerordentlich

gut besucht. In dem Tagungsbericht von Uwe Ziegenhagen erfahren Sie dazu alles Weitere. Von der TUG 2012 berichtet dagegen Stefan Kottwitz.

»Nach der Tagung« ist immer auch ein »vor der Tagung«, die im nächsten Jahr in Leipzig stattfinden wird. Die entsprechende Einladung finden Sie in dieser Ausgabe. Ebenso wie die DVD TEX-Collection, die dieses Mal von der TUG produziert wurde und als Luftfracht in Leipzig eintraf, als die letzte Ausgabe von »Die TEXnische Komödie« gerade in der Auslieferung war.

Ich wünsche Ihnen wie immer viel Spaß beim Lesen und verbleibe

mit TEXnischen Grüßen,

Ihr Herbert Voß

Hinter der Bühne

Vereinsinternes

Liebe Mitglieder,

die letzte Mitgliederversammlung ist gefühlsmäßig grade vorbei, prachtvolles Wetter, tolle Organisation und ein interessantes Vortragsprogramm haben einen anhaltenden Eindruck hinterlassen, da steht schon die Frühjahrstagung in Leipzig vor der Tür. Die Einladung zur Tagung und Mitgliederversammlung mit dem »Call for Papers« finden Sie ebenfalls in dieser Zeitschrift. Wie angekündigt, findet dort regulär die Neuwahl des Vorstands statt. Deshalb enthält das Grußwort an dieser Stelle eine persönliche Erklärung des Vorsitzenden.

Ausgehend von der Auslastung durch meine Tätigkeit für Open Access der Konferenzpublikationen im Bereich der Beschleunigerphysik (`JACoW.org`), für die mich mein Arbeitgeber vollständig freistellt, und der damit verbundenen internationalen Reisetätigkeit, hatte ich im Sommer überlegt, mit den Neuwahlen 2012 in Leipzig aus dem Vorstand auszuscheiden.

Ich habe dies meinen Vorstandskollegen bei der Tagung in Garmisch-Partenkirchen mitgeteilt, da mein Empfinden war, dass ich durch die berufliche Tätigkeit, die immer mehr auch Teile der Freizeit in Beschlag nimmt, nicht mehr in der Lage bin, effektiv für DANTE e.V. tätig zu sein. Für mich steht in der Vorstandsarbeit nicht die Verwaltung des Vereins im Vordergrund, sondern das Einbringen und Unterstützen neuer Ideen aus der Mitgliederschaft, aus anderen TEX-Usergroups und diese neuen Entwicklungen zu begleiten und bekannt zu machen. Dies funktioniert im Rahmen der Projektförderung recht gut, da wir schon in die Statuten aufgenommen hatten, dass ein schriftlicher Bericht als Artikel für »Die TEXnische Komödie« geliefert wird und der Projektstatus auf den Tagungen vorgetragen wird. In anderen Bereichen sehe ich aber Defizite, die ich in der mir verbleibenden Freizeit nicht angehen konnte.

Die Ankündigung meines völligen Rückzugs wurde nicht euphorisch aufgenommen – in nachfolgenden Gesprächen wurde ich gebeten, zumindest als Beisitzer weiterhin die internationalen Kontakte, insbesondere zu den europäischen Usergroups zu pflegen und die Projektförderung weiterhin zu betreuen. Da ich nicht aus Frust oder ähnlichen Gründen aus dem Vorstand ausscheiden wollte, sondern wegen der Arbeitsbelastung und insbesondere auch, um den Jüngeren die Gelegenheit zu geben, neuen Wind in den Verein zu bringen, habe ich nach einiger

Überlegung zugestimmt, dafür zur Verfügung zu stehen (so man mich wählt). So weit zum persönlichen Teil.

Zum Ende des Jahres mehren sich wieder die Austrittserklärungen. Einige Zusendungen enthalten auch Begründungen, was dem Mitglied missfällt und warum es den Einsatz des Vereins bei der Förderung der Weiterentwicklung und der Verbreitung von TEX und seinen moderneren Kollegen nicht mehr unterstützen möchte.

Wir greifen an dieser Stelle einen Grund auf und zwar, dass DANTE e.V. nichts für ConTEXt tut oder veröffentlicht. Hierzu möchten wir festhalten, dass in den vergangenen Jahren erhebliche Mittel in die Weiterentwicklung von METAPOST und LuaTEX (hier insbesondere OrientalTEX) geflossen sind, die somit vornehmlich ConTEXt MkIV zugute kamen. Es wurde auch ein Beitrag zum Hosting von `contextgarden.net` geleistet. In unserer Mitgliedszeitschrift werden alle Artikel veröffentlicht, die uns von Autoren zugesandt werden. Das Fehlen von Artikeln über ConTEXt oder andere Neuentwicklungen ist sicherlich nicht der Fehler des Vereins, sondern zeigt, dass wir mehr Autoren brauchen, die diese Themen einem breiteren Leserkreis nahebringen. Als kleinen Anreiz gibt es für Aktive dann eine Beitragssenkung für das Folgejahr.

Sicherlich ist es nicht einfach, einen Artikel, der »Spezialitäten« von ConTEXt im Satz zeigen möchte, im Format und Produktionsfluss von »Die TEXnische Komödie« unterzubringen, aber unser Chefredakteur Herbert Voß stellt sich immer wieder diesen besonderen Herausforderungen. Und so können wir auch nicht nachvollziehen, warum das Aussehen unserer Mitgliedszeitschrift mit dem »Layout einer Kirchenzeitung etwa der fünfziger Jahre …» verglichen wird (Zitat aus einer Austrittserklärung).

Abschließend wünschen wir noch allen eine schöne Weihnachtszeit sowie einen guten Rutsch ins kommende Jahr! Bitte planen Sie rechtzeitig einen Besuch bei der Frühjahrstagung in Leipzig ein und informieren sich selbst über die interessanten Neuentwicklungen der TEX-Welt und lernen Sie die LuaTEX- und ConTEXt-Entwickler persönlich kennen.

Mit freundlichen Grüßen,

Volker RW Schaa	Adelheid Grob
Vorsitzender	Stellvertretende Vorsitzende

Beschlüsse der 45. Mitgliederversammlung von DANTE e.V. am 1. Oktober 2011 in Garmisch-Partenkirchen

Manfred Lotz

Zeit:	1. Oktober 2011, 9:05 Uhr – 10:01 Uhr
Ort:	Werdenfels Gymnasium Pavillon (Raum E001) 82 467 Garmisch-Partenkirchen
Teilnehmer:	33 (anhand der ausgegebenen Stimmkarten)
Leitung:	Volker RW Schaa (Erster Vorsitzender von DANTE e.V.)
Protokollant:	Manfred Lotz (Schriftführer von DANTE e.V.)

Die Mitgliederversammlung wurde satzungsgemäß eingeladen und ist beschlussfähig.

TOP 1: Begrüßung, Tagesordnung und Vorstellung des Vorstands

TOP 1.1: Begrüßung und Tagesordnung

Volker RW Schaa begrüßt die Teilnehmer der 45. Mitgliederversammlung von DANTE e.V. in Garmisch-Partenkirchen und stellt die Tagesordnung vor:

1. Begrüßung, Tagesordnung und Vorstellung des Vorstands
   - Begrüßung und Tagesordnung
   - Vorstellung des Vorstandes
2. Bericht des Vorstands
   - Zurückliegende Tagungen
   - Kommende Tagungen
3. Sonstiges
4. Freigabe der Ausgaben von »Die TEXnische Komödie«
5. TEX Collection DVD
6. Neue deutsche Trennmuster

Die Tagesordnung wird ohne Einspruch akzeptiert.

TOP 1.2: Vorstellung des Vorstands

Bis auf Adelheid Grob (Stellvertretende Vorsitzende) sind alle derzeitigen Vorstandsmitglieder anwesend und werden von Volker RW Schaa vorgestellt: Klaus Höppner (Schatzmeister), Manfred Lotz (Schriftführer), Bernd Raichle (Beisitzer), Martin Sievers (Beisitzer), Herbert Voß (Beisitzer) und Uwe Ziegenhagen (Beisitzer).

Der Verein unterhält in Heidelberg ein Büro, das von Frau Karin Dornacher geleitet wird.

TOP 2: Bericht des Vorstands

TOP 2.1: Zurückliegende Tagungen

- 5. International ConTEXt User Meeting vom 19. 9.–24. 9. 2011 in Bassenge, Belgien.

TOP 2.2: Kommende Tagungen

- Frühjahrstagung 2012 und 46. Mitgliederversammlung von DANTE e.V. am 7. 3.–9. 3. 2012 in Leipzig.
- EuroTEX 2012 und 6. Internationale ConTEXt-Tagung am 8. 10.–12. 10. 2012 in Breskens (Niederlande). Die Tagung wird gemeinsam von der NTG und DANTE e.V. organisiert und wäre auch der ideale Zeitpunkt für die Herbsttagung 2012 von DANTE e.V.
- TUG 2011 in Trivandrum, Kerala, Indien vom 19. 10.–21. 10. 2011.
- TUG 2012 in Boston, USA vom 16. 7.–18. 7. 2012.

TOP 3: Sonstiges

TOP 3.1: Freigabe der Ausgaben von »Die TEXnische Komoedie«

Es gibt eine Diskussion darüber, wann die Ausgaben von »Die TEXnische Komoedie« ins Web gestellt werden sollen. Zur Zeit findet man alle Ausgaben der DTK, welche mindestens 3 Jahre alt sind, auf `http://www.dante.de/DTK/Ausgaben.html`. Es gibt den Wunsch, die DTK direkt für Mitglieder freizugeben.

Dazu hat Martin Schröder einen Antrag für die Frühjahrstagung 2012 in Leipzig gestellt: *Der Vorstand wird aufgefordert, dafür zu sorgen, dass die DTK deutlich zeitnaher den Mitgliedern und der Allgemeinheit zur Verfügung gestellt wird. Ich verweise hierbei auf die Umsetzung bei TUG und NTG*

Mit dem Verweis auf die Umsetzung bei TUG und NTG ist der geschützte Benutzerbereich gemeint, der jeweils bei TUG und NTG existiert. Dort können die

jeweils neuesten Ausgaben der TUG Boat beziehungsweise des NTG Magazins von Mitgliedern heruntergeladen werden.

Der Vorstand tendiert dazu, alle Ausgaben der DTK, die mindestens 1 Jahr alt sind, auf die TEX Collection DVD zu packen.

TOP 3.2: TEX Collection DVD

Die TEX Collection DVD 2011 wurde zum ersten Mal von der TUG produziert und traf zwei Tage zu spät ein, um der 3. Ausgabe der DTK beigelegt werden zu können. Sie wird nun mit der 4. Ausgabe versandt.

TOP 3.3: Neue deutsche Trennmuster

Bernd Raichle berichtet, dass es neue deutsche Trennmuster gibt, die weniger Trennfehler erzeugen und die bisherigen Trennmuster ersetzen könnten. Allerdings bedeutet dies auch, dass dann die Umbrüche anders sind. Um einen Umstieg auf die neuen Trennmuster möglichst abwärtskompatibel zu gestalten, schlug Herbert Voß vor, die alten Trennmuster als Standard zu belassen und eine neue Option zu schaffen, um die neuen Trennmuster einzuschalten.

Volker RW Schaa schließt die Versammlung um 10:01 Uhr.

Volker RW Schaa	Manfred Lotz
(Versammlungsleiter)	(Protokollant)

DANTE 2012 – Einladung zur Mitgliederversammlung und »Call for Papers«

Volker RW Schaa, Gerrit Imsieke

Liebe Mitglieder von DANTE e.V., die Frühjahrstagung DANTE 2012 findet vom 7.–9. März 2012 an der Hochschule für Technik, Wirtschaft und Kultur (HTWK) in Leipzig statt. Veranstalter sind DANTE e.V. und le-tex publishing services GmbH.

Am Mittwoch sind Tutorien geplant, Donnerstag und Freitag sind für Vorträge und die 46. Mitgliederversammlung von DANTE e.V. vorgesehen. Die Tagesordnung der Mitgliederversammlung am Donnerstag, den 8. März 2012 um 9.00 Uhr in der

HTWK
Lipsius-Bau
Raum LI 318
Karl-Liebknecht-Straße 145
04277 Leipzig

lautet:

1. Begrüßung und Tagesordnung
2. Bericht des Vorstands
3. Finanzbericht
4. Bericht der Kassenprüfer
5. Entlastung des Vorstands
6. Wahl eines Vorstands
7. Wahl der Kassenprüfer
8. Abstimmung über den Antrag »Zeitnahe allgemeine Veröffentlichung von ›Die TEXnische Komödie‹ im Web«
9. Verschiedenes

Ihre Stimmunterlagen erhalten Sie direkt vor Ort, um vorherige Anmeldung wird gebeten. Eine Übertragung des Stimmrechts ist im Rahmen des § 13 (4) der Vereinssatzung möglich. Wie üblich sind auch Nichtmitglieder als Gäste willkommen.

Falls Sie ein Tutorium oder einen Vortrag anbieten wollen, werden Sie gebeten, dies mit dem entsprechenden Formular auf der Tagungsseite im Internet oder per E-Mail an `dante2012@dante.de` anzumelden. Fügen Sie hierfür bitte eine Kurzzusammenfassung (Abstract) als Text- oder TEX-Datei in einem der üblichen Formate (LATEX, XETEX, LuaTEX oder ConTEXt) bei.

Unter `http://www.dante.de/events/dante2012.html` finden Sie die Tagungsseite mit allen weiteren Informationen.

Mit Fragen, Wünschen und Anregungen wenden Sie sich bitte an

DANTE e.V.
Stichwort: DANTE 2012
Postfach 10 18 40
69008 Heidelberg

Mit freundlichen Grüßen,

Volker RW Schaa (Vorsitzender DANTE e.V.)
Gerrit Imsieke (le-tex)

TEX-Theatertage

Bericht von der 45. Mitgliederversammlung

Uwe Ziegenhagen

Der Wetterbericht verhieß schon exzellente Aussichten, als ich an diesem Freitagmorgen um kurz vor 6:00 Uhr in den ICE von Köln nach München stieg, um nach Garmisch-Partenkirchen zu fahren. Nach sechs Stunden Anreise stellte sich heraus, dass der Wetterbericht Recht behalten sollte, denn die Sonne war DANTE wohlgesonnen und bescherte uns fantastisches Wetter an Deutschlands Südspitze. Ich weiß nicht, wie die Organisatoren Andreas, Markus und Stefan dies bewerkstelligt haben, empfehle aber die gleiche Vorgehensweise bei künftigen DANTE-Tagungen.

Der erste Teil der Tagung bestand aus dem Einführungstutorium am Freitagnachmittag. Trotz des schönen Wetters fanden sich einige Zuhörer ein – die meisten TEX-erfahren – , um Ideen und Anregungen zu diskutieren.

Schon beim Vorabendtreff im »La Baita« zeigte sich, dass diese Herbsttagung außerordentlich gut besucht war. Nicht nur der Süden Deutschlands war zahlreich erschienen, auch aus Hannover und Berlin waren TEX-Enthusiasten angereist, um die bayerische Gastfreundschaft zu genießen.

Die eigentliche Tagung am Sonnabendmorgen begann mit der Registrierung und Begrüßung der Mitglieder, bei der anschließenden Mitgliederversammlung sprach der Vorsitzende von DANTE e. V. Volker R. W. Schaa einige vereinsspezifische Themen und künftige Termine an. Da aber nichts zu entscheiden war, konnte dieser Teil recht kurz gehalten werden und Herbert Voß bekam die notwendige Zeit für seinen Vortrag über das Font-Chaos von XELATEX und LuaTEX. Er zeigte, dass die Unterstützung der Linux-Systemschriften bei XELATEX/LuaLATEX eine Vielzahl toller Möglichkeiten bietet, es auf dem Weg zum perfekten Ergebnis jedoch einige Hürden zu nehmen gilt.

Arno Trautmann hielt im Anschluss einen sehr ansprechenden Vortrag zum Thema »Node-Bearbeitung mit LuaTEX«. Hinter dem für LuaTEX-Unerfahrene unscheinbaren Titel verbarg sich eine Einführung in Methoden, mit deren Hilfe man LuaTEX beibringen kann, den LATEX-Quellcode vor der Ausgabe in ein Dokument

Abbildung 1: Vorabendtreff im »La Baita«

dynamisch zu verändern. Arno zeigte, wie man mit etwas Lua-Code Textsequenzen in Regenbogenfarben setzen kann, alle Großbuchstaben einfärben oder Texte mittels ROT13 verschlüsseln kann. Für mich persönlich war dieser Vortrag der lang erwartete Einstieg in LuaTEX, denn zu mehr als dem Zusammenaddieren von zwei Zahlen waren meine Lua-Kenntnisse bisher nicht ausreichend.

Um die Abkehr vom klassischen gedruckten Buch hin zu E-Books ging es im nächsten Vortrag von Thomas Ferber. Anhand von vielen Beispielen zeigte er, wie sich Texte für E-Book-Reader umsetzen lassen und welche Eigenheiten es zu beachten gilt.

Sehr interessant war auch der folgende Vortrag von Stefan Mayer, der die notwendigen Pakete vorstellte, um mit LATEX Texte zu setzen, die den Anforderungen der wichtigsten psychologischen Fach-Gesellschaften genügen. Im anschließenden Vortrag von Uwe Lück ging es dann um die Dokumentation von Paketen und wie die notwendigen Arbeiten an den Paketdokumentationen vereinfacht werden können. Richard Reindl und Alfred Wassermann gaben eine Einführung in die SMART-Aufgabensammlung für Mathematik und Physik. Die SMART-Sammlung, in den letzten knapp 20 Jahren entwickelt, enthält mehrere tausend Schulaufgaben für Gymnasien und Realschulen mit den entsprechenden Musterlösungen. Statt sich also selbst für jede Unterrichtsstunde neue Aufgaben ausdenken zu

Abbildung 2: Tagungsessen im »3 Mohren«

müssen, kann ein Lehrer hier einfach und schnell Aufgaben bekommen, auch Schülern steht der Download der Übungsaufgaben offen.

Im letzten Vortrag des Tages demonstrierte Doris Wagner, wie man mit Hilfe von PSTricks Geometrieaufgaben des bayrischen G8-Abiturs veranschaulichen und Lösungen entwickeln kann. Die Folien zu diesem und allen anderen Vorträgen sind online abrufbar: `http://www.dante.de/events/mv45/Programm.html`.

Im Anschluss an die eigentliche Tagung ging es dann zum Tagungsessen im »Drei Mohren«. Bei bayerischen Leckereien aus Pfanne und Backofen konnte man hier die Vorträge des Tages Revue passieren lassen, alte Kontakte wiederbeleben und neue knüpfen.

Der Sonntag stand dann ganz im Zeichen des touristischen Beiprogramms. Am Garmisch-Partenkirchener Ski-Stadion, unweit der Sprungschanze, trafen sich die Tagungsteilnehmer und wanderten gemeinsam zur Partnachklamm. Für einen Flachlandtiroler wie mich, der den Begriff »Klamm« nur aus dem Herrn der Ringe und gegebenenfalls als Beschreibung der finanziellen Situation zu Studienzeiten kennt, war es ein tolles Erlebnis, durch diese Schlucht zu wandern, immer bedacht darauf, einen guten Punkt für ein Foto bei gleichzeitig ausreichender Trittsicherheit zu finden. Nach dem Durchwandern der Klamm und dem anschließenden Gruppenfoto erklommen wir den nahegelegenen Berg (der für die Einheimischen sicherlich nur ein besserer Hügel ist), wo wir uns bei Käsespätzle und Germknö-

Abbildung 3: Gruppenbild hinter der Partnachklamm

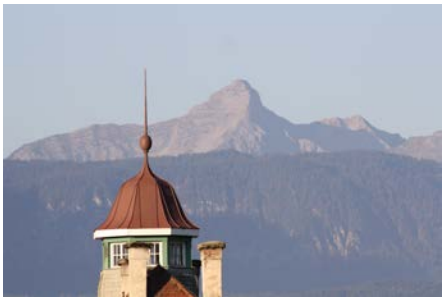
deln stärkten. Die Gruppe teilte sich dann auf, um ins Tal zurückzukehren oder noch weiter die Bergluft an diesem wunderbaren Herbsttag zu genießen.

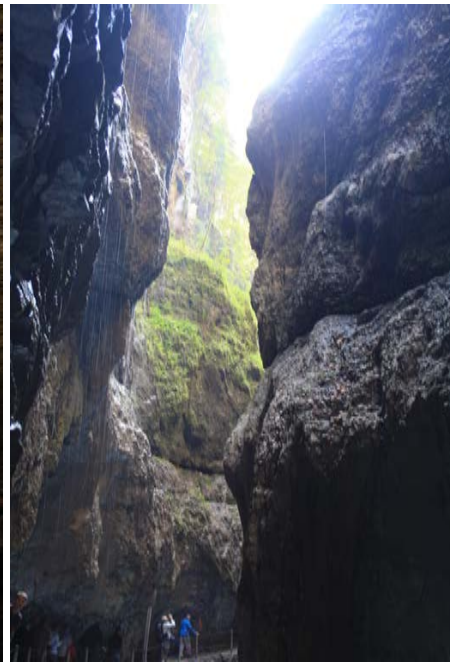
Was sicher bleibt, sind die Erinnerungen der Teilnehmer an eine tolle Tagung und der persönliche Wunsch, den Urlaub hier zu verbringen. Im Namen aller Teilnehmerinnen und Teilnehmer möchte ich den Organisatoren für die tolle Planung und Durchführung der 45. Mitgliederversammlung danken, die diese Tagung (sicherlich auch begünstigt vom fantastischen Herbstwetter) zu einem einmaligen Erlebnis haben werden lassen.




CALIBRE





(Fotos: Reinhard Kotucha/Herbert Voß)

TUG 2011 in Thiruvananthapuram (Indien) vom 19. bis 21. Oktober

Stefan Kottwitz

Die diesjährige Konferenz der TEX Users Group fand vom 19. bis 21. Oktober in Thiruvananthapuram statt. Dieser Ort mit dem schwierigen Namen hieß früher Trivandrum und ist die Hauptstadt des indischen Bundesstaats Kerala. Ursprünglich war Ägypten als Veranstaltungsort für den Sommer geplant, doch der aktuellen politischen Unruhen wegen wurde der Tagungsort verlegt. Die in Indien ansässige Firma River Valley Technologies (RVT) sprang kurzfristig ein und organisierte dieses 32. TUG-Meeting.

Ich plante schon länger, Mitglied der TUG zu werden und auch mal eine Konferenz zu besuchen. Ersteres ergab sich für mich, da ich im Juli 2011 »Stack Exchange Inc.« anregte, institutionelles Mitglied der TUG zu werden, was im September dann umgesetzt wurde. Hintergrund ist, dass »Stack Exchange« eine TEX Webseite hostet (`http://tex.stackexchange.com`), bei der ich als Moderator aktiv bin. Damit sind acht individuelle Mitgliedschaften verbunden, die an aktive Nutzer der Seite verteilt wurden, so auch an mich. Weiterhin sponsorte mir »Stack Exchange« die Teilnahme an der Konferenz, so dass mir die Reise in das weit entfernte Südindien leicht fiel.

Montag flog ich von Hamburg über Dubai und Mumbai nach Trivandrum, wo mich bei Ankunft am Dienstag ein Transfer von RVT erwartete und mich zum Hotel fuhr. Ich wohnte im Somatheeram Ayurveda Resort, sehr schön mit einem großartigen Ausblick zum Ozeanstrand. Der größte Teil der Teilnehmer wurde im benachbarten Manatheeram Resort untergebracht. Am Mittwoch sollte es losgehen.

Der Tagungsort

Die Konferenz fand auf dem RVT-Campus statt. Die Vorträge und Diskussionen wurden in einem klimatisierten Konferenzraum abgehalten, mit Stromversorgung an jedem Platz sowie WiFi für die Teilnehmer. Projektor und Audio-Technik gab es naürlich, und mehrere Kameras zeichneten die Vorträge auf, sowohl für ein Live-Streaming ins Internet als auch die Sprecher und das Leinwandbild für eine spätere Veröffentlichung.

Das Mittagessen nahmen wir draußen in einem großen Zelt ein; es gab traditionelles indisches Essen vom Buffet, was hervorragend schmeckte.

Für gelegentliche Gespräche traf man sich auch draußen unter einem schattenspendenden Dach, traditionell mit Kokospalmen-Blättern gedeckt. Um diese Zeit war es in Indien übrigens richtig heiß, mit tagsüber 30 Grad und mehr.

Der Campus ist etwa 45 Fahrminuten von den Hotels entfernt, RVT sorgte für tägliche Transfers zwischen Campus und Hotels mit Bussen und Autos.

Die Teilnehmer

Fünfzig Teilnehmer hatten sich registriert. Dreiundzwanzig kamen aus Indien, jeweils vier aus Deutschland und aus Großbritannien, drei aus den USA, auch Frankreich, Tschechien und Australien waren vertreten, eine bunte Mischung also. Viele mir namentlich bekannte TEX-Freunde lernte ich hier erstmals kennen. Die deutschen Teilnehmer waren Reinhard Kotucha, Frank Mittelbach, Thomas Ratajczak und ich.

Das Programm

Der Zeitplan war an allen drei Tagen gleich: Vormittags gab es vier Vorträge mit einer Kaffeepause dazwischen, dann Mittagessen und am Nachmittag vier Vorträge mit einer Pause darin.

Einige Themenschwerpunkte waren die Verwendung von TEX mit anderem Markup wie XML, HTML und HTML5, CSS, MathML und EPUB, fortgeschrittene PDF-Verwendung, weiterhin wurde typographisches Design besprochen.

Tag Eins

Barbara Beeton eröffnete die Konferenz mit einigen Worten, dann sprach Ross Moore über Fortschritte in der Verwendung von Mathematik in »tagged« PDF-Dokumenten. Während gewöhnliche PDF-Dokumente Inhalte wie Text, Grafiken, Hyperlinks und Lesezeichen enthalten, verfügen »tagged« PDF-Dokumente zusätzlich über logische Struktur. Diese Struktur kann neben Gliederungsinformationen auch die Anordnungsreihenfolge, Absatz- und Bildattribute enthalten. Das erlaubt beispielsweise angepassten Lesefluss bei elektronischen Büchern sowie barrierefreie Dokumente. Zum Beispiel kann man mit Struktur angereicherte Dokumente vom Computer vorlesen lassen, etwa wenn man schlecht oder gar nicht sehen kann. Ross Moore präsentierte das eindrucksvoll, indem er einen komplexen mathematischen Text vorlesen ließ, wo man auch Indices, Klammerungen und Matrizen sehr gut einsetzen konnte. Hierfür wurde TEX-Code mit MathML-Beschreibungen vereint, was sich automatisieren lässt. Dennoch scheint der Satz solcher Dokumente enorm komplex zu sein. TEX wurde bislang kaum für barrierefreie Dokumente oder solche strukturierten PDFs verwendet, das kann sich ändern, wenn dieser Anspruch bei Verlagen entstehen sollte.

In der nächsten Präsentation zeigte uns T. Rishi, wie man mit Ebenen in PDF arbeiten kann. Verschiedene Versionen des gleichen Dokuments lassen sich hier ebenenweise aus- und einblenden, was zum Korrekturlesen nützlich ist, vergleichbar mit dem klassischen Übereinanderhalten zweier Ausdrucke gegen das Licht. Weiterhin zeigte er die Anwendung von Tooltips für Korrektur-Arbeiten, zum Zitieren und Referenzieren von Tabellen und Abbildungen. Es ist beeindruckend, wenn man den Mauszeiger auf eine Referenz zieht und die zugehörige Abbildung wird eingeblendet, und sie verschwindet wieder, wenn man den Zeiger weiterbewegt.

Nach einer Pause mit Kaffee und Tee hielt C. V. Radhakrishnan eine Präsentation über TEX4ht, ein hochentwickeltes Tool zur Umwandlung von TEX nach HTML und XML einschließlich MathML. Das erlaubt beispielsweise den Export nach MediaWiki, MathJax, Braille, Sprachausgabe oder das Auflösen von LATEX-Makros. Es gibt verschiedene Ansätze: LATEX2HTML, LATEXML und Tralics parsen LATEX-Code

und schreiben XML. Ein anderer Weg ist variables Markup, was nachbearbeitet werden kann, beispielsweise durch XSLT, das entweder XML oder LaTeX ausgibt. TeX4ht geht anders vor: Während des Textsatzes fügt es XML-Elemente durch DVI-`\special` ein, um die DVI-Dateien weiterzuverarbeiten. Durch die Verwendung des TeX-Compilers können bei der Umwandlung sogar komplexe Autor-Makros verarbeitet werden, was ein Problem ist, wenn man direkt vom LaTeX-Quellcode aus umwandelt. Ross Moore erläuterte von TeX4ht bereitgestellte Hooks, die man für die meisten LaTeX-Befehle verwenden kann, wie beispielsweise am Anfang und am Ende von Abschnitten sowie auch von deren Überschriften.

Karel Skoupý sprach über die Anwendung von TeX als Programmiersprache, insbesondere mit Bezug auf ε-TeX. Er demonstrierte Implementierung und Anwendung von Records und Objekten, mit Ausblick auf Wiederverwendung und Vererbung, Namensräume, bis hin zu virtuellen Methoden und multipler Vererbung. Da stellt sich die Frage, warum man nicht einfach eine geeignete Programmiersprache nimmt und diese mit TeX kombiniert. Im Falle von Karel Skoupy ist es offenbar einfach das Vergnügen, es mit TeX zu tun.

Nun war es Mittagszeit und es gab ein großartiges Buffet mit Spezialitäten aus nord- und südindischer Küche. Die Mittagspause war lang genug, um sich miteinander am Mittagstisch oder an schattigen Plätzen der Umgebung zu unterhalten.

Den Nachmittag eröffnete Kaveh Bazargan, er sprach darüber, warum TeX heute noch wichtiger sei als früher. Er begründete, dass auch in Zeiten von WYSIWYG-Textverarbeitung heute Markup und Quellcode-Verfügbarkeit sehr wichtig sind. Auch in Zeiten von HTML und XML ist TeX-Markup immer noch herausragend, etwa wenn man die Lesbarkeit mit XML-Markup vergleicht. Das ließ sich in fünf Minuten sagen, und stieß natürlich auf große Zustimmung der anwesenden TeX-Freunde.

Im Anschluss führte Alan Wetmore die Funktionalität und Handhabung verschiedener E-Book-Lesegeräte vor und diskutierte Vor- und Nachteile von EPUB- und PDF-Format. Er zeigte, wie LaTeX für E-Reader-Ausgabe verwendet werden kann. Für die zu erwartenden Probleme mit Weißraum und Seitengeometrie-Änderungen (PDF-Reflow) stellte Alan Wetmore Lösungen vor, die er auf verschiedenen Geräten per Projektor vorführte.

Boris Veytsman zeigte uns dann, wie man die Ausgaberoutinen von TeX und LaTeX einfach anpassen kann, was manchmal als schwierig bezeichnet wird. Ziel war die Ausgabe auf einem E-Book-Reader. Grundsätzlich stellt sich hier die Frage, welche Eigenschaften man von Büchern übernimmt und welche nicht. Boris Veytsman schlug vor, den Buchtext einfach wie auf einer langen vertikalen Rolle zu lesen. So brauchen wir keine Seitenumbrüche, was bedeutet: Es besteht kein Bedarf an Gleitobjekten und Positionierungsfragen, keine Witwen und Waisenkinder!

Warum auch sollten wir Seiten auf einem E-Reader nach rechts »umblättern«, statt einfach nach unten zu scrollen? Beim Kopieren solcher physischer Eigenheiten müssen E-Reader mit deren Nachteilen kämpfen. Boris Veytsmans schlägt hier die Implementierung variabler Seitengrößen in TEX und in LATEX vor. Dafür zeigte er einfache Hacks der Ausgabe-Routinen, einmal durch Manipulieren von `\pdfpageheight` in pdfTEX sowie durch `everyshi` and `\EveryShipout` mit LATEX. Fragen, wie man dann mit Seitenzahlen und Fußnoten umgehen könnte, wurden angesprochen und noch etwas offen gelassen.

Am Ende des Programms des ersten Tages hörten wir noch einmal Rishi, der uns Tools und Workflows demonstrierte, wie man von TEX ausgehend mittels TEX4ht zu strukturiertem TEX und XML kommt und dann diese weiter zu PDF oder EPUB umwandeln kann.

Anschließend brachte uns ein Bus zu den Hotels zurück, wo man das Strand-Resort genießen konnte, oder wie ich das Hotel-WiFi nutzte – ich notierte tägliche Berichte der Tagung auf `http://tex.blogoverflow.com`.

Tag Zwei

Das Programm begann mit Jean-Michel Hufflen, der in seinem Vortrag verschiedene Wege zur Erzeugung von Bibliografien verglich. Zunächst stellte er die klassischen Tools vor, wie Tıb und BibTEX, um dann auf aktuelle Probleme zu sprechen zu kommen, betreffend Mehrsprachigkeit und Vermischung von Eingabecodierungen. Der Fokus der Präsentation lag auf den BibTEX-Nachfolgern, wie biblatex, wo die Formatierung mit LATEX-Mitteln erfolgt, biber, Bibulus und CrossTEX, und insbesondere MIBibTEX, was man sich als BIBTEX, erweitert um multilinguale Features, vorstellen kann, mit XML als Grundlage und einer neuen Sprache für bibliografische Stile.

Brian Housley stellte die von ihm entwickelte neue Briefklasse »hletter« vor, eine Erweiterung der Standard-Briefklasse, die mit zusätzlichen Features mehr Komfort bieten soll. Im November erschien sie auf CTAN.

Dann kam eine besonders umfangreiche Präsentation, Didier Verna sprach über Programmierstandards in LATEX. Er begründete, dass ein guter und vor allem konsistenter Programmierstil enorm wichtig ist, damit Code auch zukünftig gelesen, verstanden und gewartet werden kann. Didier Verna gab eine Menge persönlicher Tipps, um gut lesbaren Code zu schreiben: Angefangen bei Einrückungen bis hin zu konsistenter Benennung von Makros, Interaktionen mit anderen Paketen, beispielsweise Namenskonfliktvermeidung durch Präfixe statt durch Allerweltsnamen.

Den Vormittag schloss Frank Mittelbach mit einer Darstellung der LaTeX3-Architektur und zu dem aktuellen Stand der Entwicklung.

Nach der Mittagspause folgte ein Gespräch über Skype mit Dave Crossland, der als Consultant für Google an Web-Fonts gearbeitet hat und uns dieses Projekt vorstellte, sowie das FreeType-Projekt.

Boris Veytsman and Leyla Akhmadeeva präsentierten dann ihr ambitioniertes Projekt, typographische Qualitäten wie Lesbarkeit messbar zu machen. In Versuchsreihen soll die Lesbarkeit von Texten verglichen werden, hierbei wird TeX als Layout-Tool eingesetzt und METAFONT zur Justierung der Schriftparameter. Das Ziel ist eine Datenbank, aus der man entnehmen kann, in welcher Weise Lesbarkeit und Verstehbarkeit von typographischen Parametern wie Schriftgröße, Groß-/Kleinschreibung, Serifen, Zeilenabstand und Zeilenlängen abhängt.

Karel Skoupy beschrieb die Gestaltung von grafisch ausgefallenen Fremdsprachenführern mit hoher Inhaltsdichte. Er zeigte Lösungswege, ein generisches Design zu schaffen, das sich für eine Vielzahl von Sprachversionen mit minimal notwendigen Anpassungen eignet. Herausforderungen sind unter anderem gemischte Alphabete, Unicode-Eingaben, asiatische Schriften, verschiedene Leserichtungen, komplexe Tabellen und das Box-Balancieren. Karel Skoupy verwendete hierzu LuaTeX mit Lua-Scripting und generischen Paketen für Records und Tabellen, die er in seiner ersten Präsentation schon ansprach. Die von ihm hierfür entwickelten Pakete sind jedoch leider proprietär, da sein Arbeitgeber Lingea bislang einer Veröffentlichung nicht zustimmte.

Schließlich sprach Dominik Wujastyk über das Setzen von Sanskrit, sowohl mit Roman als auch mit Devanagari. Während die Ausgabe mit XeLaTeX heute elegant geht, ist die korrekte Silbentrennung bei Sanskrit recht kompliziert.

Am Abend des zweiten Tages trafen wir uns alle zum Bankett. Das festliche Abendessen fand in einem Strandhotel statt, aufgrund des heißen Klimas doch eher in lockerer und bunter Kleidung. Wir saßen an Tischen im Freien, am schön beleuchteten Pool im Grünen, und wer sich jetzt noch nicht kannte, der konnte es nachholen.

Tag Drei

Ein besonderer Vortrag kam von Pavneet Arora. Er begann mit der Bedeutung von Gestaltung für den Inhalt, für Beeinflussung der Wahrnehmung und den Transport von Ideen. Er betonte, dass TeX-Anwender nicht nur eine Technologie benutzen, sondern dass sie auf diesem Wege auch etwas über Typographie und Design erfahren, und auch, dass weniger manchmal mehr ist. Pavneet Arora krönte seinen Vortrag mit der Erläuterung des Designs seines Hauses, vom goldenen

Schnitt bis hin zur im Mauerwerk verewigten Eulerschen Gleichung – das nenne ich Begeisterung.

Jean-Luc Doumont demonstrierte uns dann, wie man die völlig verschiedenen Programmiersprachen PDF und pdfTEX integrieren kann, beispielsweise TEX-Größen als Argumente für PDF-Befehle verwendet, konsistent mit Punkten und Linien in beiden Welten agiert und beispielsweise eine PDF-Rotation um einen in TEX berechneten Winkel durchführt.

Die Präsentation von Sukumar Sankar, S. Mahalakshmi und L. Ganesh zeigte uns den Ansatz einer XML-Version von CSS, die plattformübergreifend verwendet werden kann. Aus dieser XML-Stylesheet-Sprache können mit XSLT sowohl TEX-Vorlagen als auch CSS3 erzeugt werden.

S. K. Venkatesan zeigte im Anschluss die Verwendung von TEX als Markup-Sprache für HTML5.

Kannan M. Moudgalya, Professor an der IIT Bombay, erläuterte uns dann, wie die indische Regierung freie und quelloffene Software fördert, und stellte uns das Projekt »spoken tutorials« vor. Hierbei werden aus Bildschirmvideos und Sprachaufzeichnungen Lehrvideos erstellt. Vorteile dieser Tutorien sind niedrige Produktionskosten und geringe Anforderungen an die Abspielhardware, insbesondere kann hier die Audiospur durch beliebige Sprachen ersetzt werden. In lokale Sprachen und Dialekte wird jeweils von Einheimischen übersetzt, so dass Wissen möglichst leicht verbreitet werden kann. Insbesondere gibt es solche Lehrvideos auch für LATEX.

Nach dem Mittagessen hielt ich einen Vortrag über TEX-Online-Communities, also über Diskussionsgruppen und Foren und insbesondere über die Schaffung von Inhalten und abrufbarem Wissen darüber. Während Usenet-Gruppen wie `comp.text.tex` und Mailing-Listen wie `texhax` sich sehr gut zum Diskutieren eignen, wie ebenfalls Webforen, entsteht hier doch noch keine gut zugängliche Wissensdatenbank. Per Hand werden FAQs erstellt, oder man verlässt sich auf eine Volltextsuche, womit sich 25 Jahre TEX-Online-Informationen schwer erschließen lassen. `http://tex.stackexchange.com` geht hier einen neuen Weg: Auf der Basis einer Frage- und Antwortseite wird hier eine indizierte und gut strukturierte Datenbank erstellt, samt einem modernen Interface, u. a. mit automatischer Dubletten-Erkennung. Die Seite wird durch die Nutzer selbst moderiert und qualitativ sortiert, der Inhalt ist frei mit cc-wiki-Lizenz. Die gesamte Datenbank wird regelmäßig als Dump zum Download bereitgestellt. Ziel ist die Schaffung einer TEX-Wissenssammlung, deren freie Verfügbarkeit so garantiert wird.

Manjusha Joshi führte dann vor, wie mit Hilfe des Paketes SageTEX die Ergebnisse von Berechnungen des Open-Source-Mathematik-Systems Sage direkt in ein LATEX-Dokument eingebettet werden können.

Petr Sojka verglich verschiedene Mathematik-Suchmaschinen, wie MathDex, EgoMath, Springers »LaTeX-Search and LeActiveMath«. Während klassische Suchmaschinen mit Wörtern funktionieren, stellt uns die Formel-Suche auch heute noch vor Schwierigkeiten – erst recht, wenn man verschiedene Notationen und die Semantik berücksichtigt. Außerdem fehlt eine geeignete Abfragesprache. Petr Sojka stellte MIaS vor, den »Math Indexer and Searcher«, entwickelt für die »European Digital Mathematics Library«.

Zum Schluss sprach Dominik Wujastyk über die Erfahrungen in Satz und Produktion eines umfangreichen Buches, anhand der Memoiren seines Vaters.

Während viele Teilnehmer noch einige Tage in Indien verbrachten, flog ich direkt im Anschluss nach Hamburg. 13 Stunden Zwischenstopp in Bangalore eigneten sich auch gut zum Bloggen und Berichten von der Konferenz. In den Folgetagen war noch viel Verkehr auf der TUG2011-Mailingliste, vielfach mit herzlichem Dank an RVT, dem ich mich sehr anschließe: Alle Achtung und vielen Dank für die Ausrichtung dieser Konferenz.

In der TUGboat-Ausgabe 32:3 werden Beiträge der TUG2011 erscheinen, Abstracts kann man auf der TUG-Homepage einsehen (`http://tug.org/`).

Bretter, die die Welt bedeuten

Attribute und Farben[1]

Manuel Pégourié-Gonnard

Wir untersuchen eine Erweiterung von LuaTEX, die *Attribute*, und ihre mögliche Nutzung bei der Implementierung der Farben. Nachdem wir das Konzept der Attribute sowie die TEX- und Lua-Schnittstellen zu ihrer Verwendung vorgestellt haben, erinnern wir an die Grundzüge der klassischen Farbimplementierung in LATEX mit ihren bekannten Problemen. Wir untersuchen dann im Einzelnen, wie man diese Probleme mit Hilfe der Attribute löst. Dies veranschaulicht einige allgemeine Prinzipien, wie man Attribute benutzt und anwendet, was sich offensichtlich nicht auf Farben beschränkt.

Anmerkung: Dank an Heiko Oberdiek, der mir erlaubt hat, zur Veranschaulichung dieses Artikels den Code seines `luacolor`-Paketes zu benutzen.

Vorstellung der Attribute

Konzept

Um einen Text zu setzen, muss jedes Schriftzeichen einer bestimmten Schriftart entnommen werden. Wenn man sich dabei nicht zu der allergrößten Schlichtheit zwingen will, wird man an verschiedenen Stellen des Textes verschiedene Schriftarten verwenden wollen. Deshalb kennt TEX den Begriff der »aktuellen Schriftart«, um die Schriftzeichen zu einem gegebenen Zeitpunkt zu setzen.

Schriftartenwechsel unterliegen den Gültigkeitsregeln[2] von TEX, und die aktuelle Schriftart wird für jedes Schriftzeichen einzeln in dem Augenblick abgespeichert, in dem es der aktuellen Liste hinzugefügt wird und *nicht* in dem Augenblick, in

[1] Zuerst erschienen in: Cahiers Gutenberg, 54–55 (2010), S. 57–85. Deutsche Übersetzung von Patrick Gundlach, Heiko Oberdiek und Petra Rübe-Pugliese.

[2] Gemeint ist der Umstand, dass TEX-Definitionen, wenn sie nicht ausdrücklich mit einem global-Attribut versehen sind, nur innerhalb einer sogenannten Gruppe gültig sind, was oft sehr vorteilhaft ist.

dem es in dem Dokument tatsächlich benutzt wird. Dieser Unterschied ist von wesentlicher Bedeutung für das folgende Beispiel[3]

```
\setbox0=\hbox{L} \it L\unhbox0 L
```

welches » *L*LL« erzeugt: Die aktuelle Schriftart wurde für das Schriftzeichen in der Box gespeichert und wird später nicht mehr geändert.

Zum Satz von mehrfarbigem Text wäre eine entsprechende Funktionalität der »aktuellen Farbe« wünschenswert, die den gleichen Gültigkeitsregeln gehorchen und für jedes Schriftzeichen einzeln abgespeichert werden sollte. Dies war beim Entwurf von TEX nicht vorgesehen und – bis zu LuaTEX – nicht hinzugefügt worden: Die Mechanismen der Farbverwaltung von pdfTEX entsprechen nicht diesen beiden Kriterien, deren Bedeutung im Folgenden erläutert wird.

Tatsächlich fügt LuaTEX, seinen Gepflogenheiten entsprechend, keinen spezifischen Farbmechanismus hinzu, sondern schlägt im Gegenteil ein flexibles Konzept vor, das den Begriff der aktuellen Schriftart verallgemeinert: die Attribute. Die Grundidee ist einfach: Jedes Schriftzeichen – oder allgemeiner: jeder Knoten (engl. *node*) einer Liste – besitzt von seiner Konzeption her mehrere Eigenschaften: seine Schriftart, seine Farbe, ggf. eine Unterstreichung usw. Der Schriftart kommt insofern eine besondere Rolle zu, als sie direkt von LuaTEX berücksichtigt wird (z. B. wenn es die Maße eines Glyphen kennen muss). Die anderen Eigenschaften werden in den Attributen gespeichert, wobei es Sache des Programmierers ist, eine geeignete Form zu wählen und später zu nutzen, z. B. mittels Rückruffunktionen (engl. *callbacks*) in Lua.

Es gibt sehr viele Attribute, von denen jedes Einzelne zu jedem Zeitpunkt einen »aktuellen Wert« hat, der entweder eine ganze Zahl oder der spezielle Zustand »deaktiviert« sein kann. Die Attribute werden durch ihre Nummer adressiert. Beispielsweise könnte man entscheiden, dass das Attribut mit der Nummer 0 die meiste Zeit über deaktiviert ist, für unterstrichene Textstellen den Wert 1 annimmt, für doppelt unterstrichene Textstellen der Wert 2 gilt usw. Man hat also mit Hilfe von Attributen eine konkrete Darstellung der abstrakten Eigenschaft »Unterstreichungslevel« festgelegt, eine Darstellung, die zu gegebener Zeit von einer aufgerufenen Lua-Funktion benutzt werden muss, um eine Wirkung zu entfalten.

Es ist wichtig festzustellen, dass LuaTEX die Anwesenheit von Attributen und deren Werte vollständig gleichgültig sind (das ist der Hauptunterschied zwischen Schriftarten und Attributen): Seine Rolle beschränkt sich darauf, den Wert jedes

[3] Der Einfachheit und der Neutralität halber basieren die folgenden Beispiele auf Plain-TEX; sie wurden auch unter LATEX mit der `article`-Klasse getestet.

Attributes gemäß den Gültigkeitsregeln (lokale Änderungen im Innern einer TEX-Gruppe) aktuell zu halten und jedem neu erzeugten Knoten die aktuellen Werte aller aktiven Attribute anzuheften – also außer denjenigen, die aktuell den Wert »deaktiviert« haben.[4]

Wir merken an, dass die Reihenfolge, in der die Knoten erzeugt werden, nicht immer intuitiv ist: Beispielsweise wird ein Ligaturknoten sehr viel später erzeugt als die ihm zugrunde liegenden Buchstabenknoten. LuaTEX besitzt eine gewisse Anzahl von Sonderregeln für Fälle, wo es wünschenswert ist, dass ein Knoten für seine Attribute andere Werte speichert als diejenigen, die zum Zeitpunkt seiner Erzeugung in Kraft sind. Im Handbuch [8, § 2.5.1] finden sich einige Einzelheiten zu diesem Thema.

Zwischenspiel: Erkundung der Boxen in Lua

TEX bietet einen Primitivbefehl, um den Inhalt von Boxen anzuschauen: `\showbox`. Insbesondere können wir mit seiner Hilfe bestätigen, dass der Zeichensatz zusammen mit jedem einzelnen Schriftzeichen abgespeichert ist. Beispielsweise erhält man mit

```
\showboxdepth=1 \showboxbreadth=10
\tt \setbox0=\hbox{a\bf b{\it c}d}
\showbox0
```

das folgende Ergebnis in der Protokolldatei:

```
> \box0=
\hbox(6.94444+0.0)x22.62762, direction TLT
.\tentt a
.\tenbf b
.\tenit c
.\tenbf d
```

Da LuaTEX diesen Primitivbefehl nicht dahingehend erweitert, dass auch die Werte der Attribute angezeigt werden, müssen wir zu deren Untersuchung unsere eigenen Werkzeuge entwickeln.

Als Erstes erstellen wir eine vereinfachte, aber anpassbare Version von `\showbox`:

```
\def\ShowBox#1{\directlua{
  for n in node.traverse(tex.getbox(\number#1).list) do
    local t = node.type(n.id)
    texio.write_nl(t)
    if (t == 'glyph') then
      texio.write(' ' .. unicode.utf8.char(n.char))
    end
```

[4] Dies ist eine rein technische Optimierung mit dem Ziel, die Größe der Knoten im Arbeitsspeicher zu begrenzen.

```
    write_node_details(n)
  end}}
```

Dieses Makro holt sich die Box mit `tex.getbox` in Form eines Knotens vom Typ `hbox` [8, § 8.1.2.1]. Dieser besitzt ein Datenfeld `list`, welches auf den Inhalt der Box zeigt. Die Funktion `node.traverse` [8, § 4.10.1.24] ist ein Iterator, der es erlaubt, die Knoten mit einem generischen `for` zu durchlaufen [5, § 4.3.4].

Als Erstes erhält man den Typ des Knotens als Zeichenkette und schreibt ihn an den Anfang einer neuen Zeile. Bei einem Schriftzeichen (Glyphen und Schriftzeichen teilen sich den gleichen Knotentyp [8, § 8.1.2.12]) gibt man das Zeichen aus, dessen Unicode-Nummer im Datenfeld `char` des Knotens steht. Schließlich ruft man zur Ausgabe der anderen Eigenschaften eine spezialisierte Funktion `write_node_details` auf:

```
\directlua{
  function write_node_details(n)
    texio.write(' font=' .. n.font)
  end}
```

Diese Funktion werden wir im Folgenden abwandeln, um statt der Schriftart die Attribute auszugeben.

Machen wir einen ersten Test:

```
\tt \setbox0=\hbox{a\bf b{\it c}d}
\ShowBox0
```

Auf dem Bildschirm und in der Protokolldatei erhalten wir:

```
glyph a font=30
glyph b font=24
glyph c font=37
glyph d font=24
```

Jetzt sind wir gerüstet, um händisch den Inhalt einfacher Boxen untersuchen zu können. Kehren wir also zurück zur Untersuchung der Attribute.

TEX-Befehle als Schnittstelle

Die TEX-Primitivbefehle zur Behandlung von Attributen sind einfach und ähneln stark denjenigen, die für die Zähler (`\count`-Register) benutzt werden. In der Tat funktionieren die Attribute genau wie die Zähler mit der folgenden – sehr wesentlichen – Ausnahme: Der aktuelle Wert aller nicht deaktivierten Attribute wird allen Knoten[5] bei ihrer Erzeugung angeheftet, außer wenn dieser Wert $-2^{31}+1$ (d. h. −"7FFFFFFF in hexadezimaler TEX-Notation) beträgt, welches

[5] Außer möglicherweise einigen Knoten von besonderen Typen, für die der Begriff eines Attributes keinen Sinn ergäbe.

der kleinste zulässige Wert ist. (Vor LuaTEX-Version 0.37 wurden alle negativen Werte als äquivalent zu diesem Spezialwert angesehen, so dass keine negativen Zahlen als Werte von Attributen benutzt werden konnten.)

Die beiden in Frage stehenden Primitivbefehle sind die folgenden[6][8, § 2.5]:

```
\attribute ⟨16-Bit-Zahl⟩ ⟨gleich⟩ ⟨32-Bit-Zahl⟩
\attributedef ⟨Kontrollsequenz⟩ ⟨gleich⟩ ⟨16-Bit-Zahl⟩
```

Der erste dient dazu, einem durch eine Zahl zwischen 0 und $2^{16} - 1$ angegebenen Attribut-Register einen Wert zwischen $-2^{31} + 1$ und $2^{31} - 1$ zuzuweisen. Dank `\attributedef` ist es möglich, eine Kontrollsequenz zu definieren, die statt `\attribute` *X* verwendet werden kann. Mit *X* ist ein spezielles Attributregister gemeint.

Im Übrigen kann man ein Attribut überall dort verwenden, wo TEX eine Zahl oder ein Register vom Typ `\count` erwartet. Insbesondere verfügt man genau wie bei diesem über `\the`, um den Wert eines Attributes zu erhalten, und über `\showthe`, um ihn in die Protokolldatei zu schreiben.

```
\attribute0=42                \showthe\attribute0
\attributedef\attrzero=0
\attrzero=2010                \showthe\attribute0
\ifnum\attrzero>2000
  \advance\attrzero by -2000
\fi                            \showthe\attrzero
```

Bei diesem Beispiel sieht man in der Protokolldatei, dass das Attribut mit Nummer 0 der Reihe nach die Werte 42, 2010 und 10 annimmt.

Obwohl man die Attribut-Register für arithmetische Operationen nutzen könnte, sind sie zum Rechnen in TEX nicht sonderlich interessant. Dafür gibt es bereits die Zähler und vor allem Lua selbst. Typischerweise werden vielmehr gewissen Attributen in der Präambel Namen (und Rollen) zugewiesen und deren Werte im Laufe des Dokuments geändert. Ausgewertet werden diese Werte dann meistens in Lua.

Lua-Schnittstelle

Der interessanteste Teil der Lua-Schnittstelle zur Verwendung der Attribute wird in Abschnitt 4.9.2 des Handbuchs [8] beschrieben. Die wesentliche Funktion ist `node.has_attribute`. Mit ihr lässt sich prüfen, ob ein Knoten ein gegebenes Attribut besitzt und ggf. dessen Wert abfragen. Die Syntax lautet:

```
v = node.has_attribute(n, a)
```

[6] Unter LuaLATEX heißen sie `\luatexattribute` und `\luatexattributedef` [10].

wobei `n` ein Knoten und `a` das uns interessierende Attribut ist. Der Variablen `v` wird entweder `nil` (im Falle, dass dieses Attribut bei der Erstellung des Knotens deaktiviert war) oder aber der Wert des Attributes zugewiesen.

Die klassische Anwendung eines Attributes besteht darin, ihm mit TEX-Makros Werte zuzuweisen und diese Werte dann aus Lua-Callbacks heraus mit `node.has_attribute` abzufragen. Es handelt sich hierbei um eine unidirektionale Kommunikation von TEX nach Lua.

Manchmal kommt es auch vor, dass man die Attribute gewisser Knoten in Lua *nach* ihrer Erzeugung ändern möchte, damit der neue Wert in einer späteren Lua-Funktion genutzt werden kann. (Erinnern wir uns, dass LuaTEX selbst die Attribute nicht berücksichtigt; es ist *unsere* Aufgabe, sie abzufragen und entsprechende Aktionen zu implementieren.) Das ermöglichen die Funktionen `node.set_attribute` und `node.unset_attribute`.

Bevor wir den Rest der Lua-Schnittstelle untersuchen, nehmen wir unser Beispiel zur Analyse von Boxen mit der folgenden neuen Funktion wieder auf:

```
function write_node_details(n)
  for attr = 0, 1 do
      texio.write(' attr' .. attr .. '=')
      texio.write(tostring(node.has_attribute(n, attr)))
  end
end
```

Modifizieren wir unser Beispiel, um Attribute statt Schriftarten zu bearbeiten:

```
\attribute0=0
\setbox0=\hbox{A\attribute1=2 B{\attribute0=-"7FFFFFFF C}D}
\ShowBox0
```

Dann können wir im Log lesen:

```
glyph A attr0=0 attr1=nil
glyph B attr0=0 attr1=2
glyph C attr0=nil attr1=2
glyph D attr0=0 attr1=2
```

In diesem Beispiel interessiert man sich für zwei bestimmte Attribute mit den Nummern 0 und 1. Das ist der häufigste Fall: Nehmen wir unsere Hypothese wieder auf, dass Attribut 0 benutzt wird, um den Unterstreichungslevel darzustellen. Offensichtlich interessieren sich Lua-Funktionen, die Unterstreichungen verwalten, nur für dieses spezielle Attribut.

Andererseits könnte man alle Attribute eines gegebenen Knotens betrachten wollen, zum Beispiel, um eine verbesserte Version von `\showbox` zu konstruieren. Man muss in diesem Fall von einer Schnittstelle auf einer niedrigeren Ebene Gebrauch machen: Die Attribute sind mit dem Knoten in Form einer Liste von

Pseudo-Knoten in dem Datenfeld `attr` des Knotens verknüpft. Man kann sie mit `node.traverse` durchlaufen, aber man muss aufpassen. Der erste Pseudo-Knoten dieser Liste markiert nur den Anfang der Liste und enthält im Gegensatz zu den folgenden selbst keine Attributdaten. Den nächsten Knoten erreicht man über das Datenfeld `next`:

```
function write_node_details(n)
  for pseudo_node in node.traverse(n.attr.next) do
      texio.write(' attr', pseudo_node.number)
      texio.write('=', pseudo_node.value)
  end
end
```

Testen wir unsere letzte Version dieser Funktion:

```
\attribute314=0
\setbox0=\hbox{A\attribute10000=1 B{\attribute314=-"7FFFFFFF C}D}
\ShowBox0
```

Das Ergebnis sieht im Log wie folgt aus:

```
glyph A attr0=0 attr314=0
glyph B attr0=0 attr314=0 attr10000=1
glyph C attr0=0 attr10000=1
glyph D attr0=0 attr314=0 attr10000=1
```

Der Vollständigkeit halber bemerken wir abschließend, dass aus Lua heraus mit Hilfe des Arrays `tex.attribute` und der Funktionen `tex.getattribute` und `tex.setattribute` der aktuelle Wert der Attribut-Register abgefragt und sogar geändert werden kann. Wahrscheinlich wird man sich dafür weniger interessieren, wenn man dazu normalerweise die TEX-Schnittstelle benutzt, während man sich in Lua eher für die Attribut-Anhaftung eines gegebenen Knotens als für die aktuellen Werte der Attribut-Register interessieren wird.

Wichtige Erweiterungen

In den oben beschriebenen Beispielen haben wir die Attribut-Nummern immer direkt verwendet und beispielsweise vorgeschlagen, Attribut 0 zur Kodierung der Unterstreichung einzusetzen. Es versteht sich von selbst, dass das bei verschiedenen, nebeneinander eingesetzten Paketen nicht funktionieren kann. Wie soll man entscheiden, zu welchem Paket Attribut 0 gehört?

Dieses Problem besteht seit Langem für die anderen Ressourcen, die TEX dem Programmierer zur Verfügung stellt: Register für Zähler, Längen, Boxen usw. Die Lösung ist ebenso lange bekannt und besteht in der Benutzung eines Makros, das die Registernummern zuteilt. Der Programmierer verwendet nur noch den symbolischen Namen, der zu dieser Nummer äquivalent definiert wurde. Plain-

TEX und LATEX stellen derartige Makros (`\newcount` u. s. w.) für die Ressourcen von TEX82 bereit.

Für die neuen Ressourcen von LuaTEX müssen wir auf ein Paket zurückgreifen, das ein solches Makro definiert: Zurzeit leisten das `luatex` und `luatexbase-attr`, die auch unter Plain-TEX eingesetzt werden können. (Die beiden stehen gegenwärtig in Konflikt, da sie dieselben Ressourcen verwalten wollen. Sie sollten in nächster Zeit zusammengelegt werden, was dieses Problem lösen würde.)

Mit dem Makro `\newluatexattribute` wird ein neues Attribut angelegt und bekommt einen Namen. Beispielsweise reserviert

```
\newluatexattribute\soulattr
```

ein Attribut für die Unterstreichung. Welche Nummer dieses Attribut tatsächlich hat, ist ohne Bedeutung; man greift darauf aus TEX heraus mit dem Befehl `\soulattr` zu:

```
\soulattr=42
\setluatexattribute\soulattr{6 * 7}
```

Die erste Zeile benutzt die einfache Syntax, die zweite eine mehr LATEX-artige Syntax, die von `luatexbase-attr` bereitgestellt wird; sie akzeptiert zusätzlich als Argument einen beliebigen von `\numexpr` interpretierbaren Ausdruck. Schließlich kann ein Attribut mit dem Makro `\unsetluatexattribute` deaktiviert werden, indem ihm ein zu der benutzten LuaTEX-Version passender Spezialwert zugewiesen wird.

Leider verlangt die Lua-Schnittstelle zur Behandlung der Attribute, dass auf die Attribute mit ihrer Nummer verwiesen wird. Zwar kann man auf den aktuellen Wert eines Attributes in symbolischer Weise über die Tabelle `tex.attribute` zugreifen; beispielsweise ist nach der vorangegangenen Zuweisung `tex.attribute.soulattr` gültig und man kann die Zeichenkette `"soulattr"` als Argument für `tex.setattribute` und `tex.getattribute` verwenden. Aber das gilt (noch) nicht für die wichtigsten Funktionen wie `node.has_attribute`.

Zum Glück erlaubt `luatexbase-attr` auf Attribut-Nummern in der Tabelle `luatexbase.attributes` mit ihrem Namen zuzugreifen. Der Lua-Code kann also folgendermaßen vorgehen:

```
-- am Anfang, aber nach \newluatexattribute\soulattr
local soulattr = luatexbase.attributes.soulattr
-- weiter hinten
local v = node.has_attribute(n, soulattr)
```

Auf diese Weise braucht der Programmierer die Nummer des benutzten Attributes nie direkt zu kennen.

Die klassische Implementierung der Farben und ihre Probleme

Warnung: Dieser und der folgende Abschnitt betreffen ausschließlich LaTeX, das einzige Gebiet, in dem der Autor den existierenden Code kennt. Wir konzentrieren uns hier auf das `color`-Paket, dessen Gebrauch als bekannt vorausgesetzt wird (ggf. beziehe man sich auf [2]). Wir beginnen, indem wir an seine Funktionsweise erinnern, wobei wir uns auf die wesentlichen Aspekte beschränken. Leser, die sich tiefer in die Materie einarbeiten möchten, können mit Gewinn in [1] und [11] (was nicht nur `color.sty`, sondern auch `graphics.sty` betrifft) nachschlagen.

Das Prinzip

Obwohl man mit Recht die Modularität von LuaTeX lobt, wäre es ungerecht zu vergessen, dass schon TeX seinen eigenen Erweiterungsmechanismus vorgesehen hatte: die *Whatsits*[7]-Knoten. Übrigens hat Knuth die Primitive `\write` (und andere) als Erweiterung programmiert, um ein Beispiel zu geben (tex.web 1340). Es gibt verschiedene Untertypen von Whatsits, die mit verschiedenen Erweiterungen (neuen Primitiven) einhergehen.

In den Kindertagen von TeX war das einzige verfügbare Ausgabeformat DVI, das seinen eigenen Erweiterungsmechanismus besaß, die `\special`-Befehle. Auch dort werden verschiedene `\special`-Untertypen zu verschiedenen Zwecken eingesetzt. Komplizierter wird es dadurch, dass `\special`-Befehle für verschiedene DVI-Interpreter (dvips, dvipdfm usw.) bestimmt sind, die nicht denselben Befehlssatz kennen. Aus TeXs Sicht bleiben die Dinge allerdings sehr einfach: `\special`-Befehle werden unverändert in das DVI eingefügt, ohne dass versucht wird, sie zu interpretieren. Sie entsprechen (ohne Überraschung) einem besonderen Untertyp von Whatsit-Knoten.

Später hat Hàn Thê Thành pdfTeX entwickelt, eine TeX-Variante, die es erlaubt, ohne Umwege PDF-Dateien zu erzeugen und aus TeX heraus auf zahlreiche spezifische Funktionalitäten dieses Formats zuzugreifen. Auch dort werden viele der neuen, mit dem PDF-Ausgabeformat verknüpften Befehle mittels Whatsit-Knoten implementiert. Zum Beispiel ist der Primitivbefehl `\pdfliteral` in seiner Philosophie dem `\special`-Befehl sehr nahe verwandt: Er erzeugt einen Whatsit-Knoten von einem geeigneten Untertyp, um Rohmaterial aufzunehmen, das in das PDF eingefügt wird, ohne dass sich TeX um den Sinn dieses Materials kümmert.

[7] Es scheint schwierig zu sein, für diesen Ausdruck eine einheitliche Übersetzung zu finden. Jean-Côme Charpentier nennt sie in seiner französischen Übersetzung des *TeXbook* ≪éléments extraordinaires≫ – außergewöhnliche Knoten.

Manuel Pégourié-Gonnard bezeichnet sie vorzugsweise als ≪nœds bidules≫ – »Dingsbums-Knoten«. Ich bleibe hier bei »Whatsit«.

So muss der Farbverwaltungscode in LaTeX mit verschiedenen Ausgabeformaten zurechtkommen: PDF direkt mit pdfTeX und im Wesentlichen einer DVI-Variante über einen DVI-Interpreter.

Um die Dinge noch ein wenig komplizierter zu gestalten, bemerken wir, dass verschiedene Methoden der Farbbeschreibung existieren: Monochromie (Nuancen einer einzigen Farbe, meistens Schwarz), additive RGB-Synthese (drei Primärfarben: Rot, Grün, Blau), CMYK-Vierfarbendruck (Cyan, Magenta, Gelb, Schwarz), benannte oder indizierte Farben (zum Beispiel arbeitet die englische Ausgabe des LaTeX-Begleiters mit genau zwei Farben: Schwarz und einem bestimmten Blau, die physisch zwei verschiedenen Druckfarben entsprechen, welche beim Drucken nicht gemischt werden).

Diese Farbmodelle werden von den Ausgabeformaten auf verschiedene Weisen unterstützt, und es ist wünschenswert, dem Nutzer eine gewisse Anzahl von Basis-Farbmodellen zur Verfügung zu stellen, die mit der Mehrzahl der Ausgabeformate genutzt werden können, auch wenn dabei intern Konvertierungen ausgeführt werden müssen.

Um mit diesen Problemen umzugehen, führt das `color`-Paket den Begriff des *Treibers* (*driver*) ein: Für jedes Ausgabeformat existiert ein Treiber, dessen Aufgabe es ist, die für das betreffende Format spezifischen Makros bereitzustellen. Nur die formatunabhängigen Makros werden in `color.sty` definiert. Die treiberabhängigen Makros haben verschiedene Aufgaben: `\color@`⟨*Modell*⟩ (zum Beispiel `\color@rgb`) nimmt als Argument eine Farbbeschreibung in einem gewissen Farbmodell in LaTeX-Syntax (zum Beispiel drei durch Komma getrennte Zahlen zwischen 0 und 1) und überführt sie in eine Beschreibung derselben Farbe (in demselben oder einem anderen Modell), angepasst an das Ausgabeformat (Beispiele siehe unten). Das Ergebnis wird in einem Makro nach Wahl abgespeichert (im Allgemeinen `\current@color`).

`\set@color` erzeugt eine an das Ausgabeformat angepasste Anweisung, um die in `\current@color` abgespeicherte Farbe zu aktivieren (im Allgemeinen durch einen Aufruf eines der vorangegangenen Makros). Es merkt auch mit einem `\aftergroup` die Ausführung von `\reset@color` am Ende der Gruppe vor.

`\reset@color` erzeugt eine an das Ausgabeformat angepasste Anweisung, um die Wirkung des vorangegangenen `\set@color` rückgängig zu machen.

`\set@page@color` erzeugt eine an das Ausgabeformat angepasste Anweisung, um aus der in `\current@color` enthaltenen Farbe bis auf Widerruf die Hintergrundfarbe zu machen. Da LuaTeX auf dieser Ebene keine interessante Neuerung anzubieten hat, werden wir uns hier nicht weiter mit diesem Makro beschäftigen.

`\define@color@named` erzeugt eine Anweisung, um in der Ausgabedatei (DVI) eine benannte Farbe zu definieren. Achtung! Dies hat nichts mit dem Makro `\definecolor` zu tun, welches einen Farbnamen in der TEX-Eingabedatei definiert. Diese Anweisung ist spezifisch für das Farbmodell *named*. Wir werden hier nicht weiter darauf eingehen, weil die Punkte, die uns interessieren, vom Farbmodell unabhängig sind.

Dies alles kann auf den ersten Blick recht abstrakt wirken. Wir werden deshalb die von `A\textcolor{red}{B}C` mit verschiedenen Treibern erzeugten Knoten ausführlich beschreiben und angeben, welche der oben genannten Makros für die einzelnen Elemente verantwortlich sind. Beginnen wir mit dvips (das Ausgabeformat ist das von `\showbox`).

```
.\T1/lmr/m/n/10 A
.\special{color push rgb 1 0 0}
.\T1/lmr/m/n/10 B
.\special{color pop}
.\T1/lmr/m/n/10 C
```

Wir weisen darauf hin, dass die Farbe `red` unabhängig vom Treiber als die Farbe `1,0,0` im RGB-Modell definiert ist. Ein Aufruf des Makros `\color@rgb` hat dies in eine an das Ausgabeformat angepasste Beschreibung umgewandelt: `rgb 1 0 0`. In diesem Fall handelt es sich um eine rein syntaktische Umwandlung, aber auch ausgefeiltere Änderungen wären möglich. Das Makro `\set@color` hat mit dieser Farbbeschreibung die erste obige `\special`-Zeile erzeugt, und von `\reset@color` stammt das zweite `\special`. Wir bemerken, dass dvips einen Farb-Stack-Mechanismus implementiert, mit dem automatisch die vorhergehende Farbe wiederhergestellt werden kann.

Schauen wir, wie es bei pdfTEX aussieht:

```
.\T1/lmr/m/n/10 A
.\pdfcolorstack 0 push {1 0 0 rg 1 0 0 RG}
.\T1/lmr/m/n/10 B
.\pdfcolorstack 0 pop
.\T1/lmr/m/n/10 C
```

Hier hat `\color@rgb` die Farbdefinition von Rot in `1 0 0 rg 1 0 0 RG` umgesetzt, das von `\set@color` benutzt wird, um den ersten Aufruf von `\pdfcolorstack` zu erzeugen, während der zweite von `\reset@color` vorgenommen wurde.

Merken wir uns aus diesem Abschnitt die drei wesentlichen (und von dem letztendlich benutzten Treiber unabhängigen) Merkmale dieser Farbimplementierung, im Gegensatz zu der mit Hilfe von Attributen realisierten Lösung, die Gegenstand des folgenden Abschnitts sein wird:

1. Die Synchronisierung zwischen Farben und TEX-Gruppen erfolgt nicht auf natürliche Weise, sondern wird durch ein `\aftergroup` erzwungen.
2. Die Farbe ist keine Eigenschaft individueller Knoten, sondern eher von Abschnitten.
3. Anfang und Ende dieser Abschnitte werden von Whatsit-Knoten gebildet (wobei der Untertyp vom Treiber abhängt).

Die Probleme

Betrachten wir als Erstes die Synchronisation mit den Gruppen: Auf den ersten Blick scheint `\aftergroup` das Problem auf elegante Weise zu lösen. Eine Einzelheit muss jedoch erwähnt werden: Der Whatsit-Knoten, der das Ende der Farbzone markiert, wird *nach* dem Ende der Gruppe eingefügt. In den meisten Fällen stört das nicht, aber wenn die in Frage stehende Gruppe von einer Box gebildet wird, ist das kritisch. Wenn man nicht aufpasst, gehört der wesentliche Knoten nicht zu der Box. Nehmen wir beispielsweise den folgenden Code:

```
Schwarz am Anfang.
{\color{red} Rot.
\setbox0\hbox{\color{green} Box}%
Davor, \usebox0, danach.}
Wieder schwarz.
```

Können Sie die Farben der Wörter »Davor« und »danach« erraten, wenn wir annehmen, dass die in der ersten Zeile aktive normale Schriftfarbe Schwarz ist? Entwickeln wir den Code: `\color{red}` erzeugt einen Whatsit-Knoten, der Rot zur aktuellen Farbe macht. Dann erzeugt `\color{green}` einen Whatsit-Knoten, der Grün aktiviert und in der Box 0 abgespeichert wird. Nach dem Ende der Gruppe (der Box) wird `\reset@color` ausgeführt, welches einen Whatsit erzeugt, der die vorhergehende Farbe wiederherstellt und nicht in die Box 0 eingefügt wird, sondern unmittelbar hinter »Rot.« Das Wort »Davor« ist deshalb schwarz. Wenn TEX die Box 0 einfügt, fügt es insbesondere den Whatsit ein, der auf Grün umschaltet. Das Wort »Box« ist also sehr wohl grün. Da es keinen Whatsit gibt, der am Ende der Box die Farbe zurücksetzt, bleibt das Wort »danach« grün. Am Ende der Gruppe wird das von `\color{red}` abgesetzte Farbwiederherstellungs-Whatsit eingefügt, und alles wird wieder normal. Dieses Beispiel wurde bewusst harmlos gewählt. Andere, nach demselben Prinzip konstruierte Beispiele rufen leicht Probleme mit pdfTEX selbst hervor, indem sie dauerhaft den Farb-Stack durcheinanderbringen, was zu Meldungen wie der folgenden führt:

```
pdfTeX warning: pdflatex: pop empty color page stack 0
```

In dem eben beschriebenen Beispiel enthält das Makro `\current@color` zu den Zeitpunkten, in denen TEX die Wörter »Davor« und »danach« setzt, sehr wohl eine

Darstellung der Farbe Rot, also genau der erwarteten Farbe. In der Tat wird dieses Makro den gewohnten Gültigkeits-Mechanismus (Gruppen) von TEX unterworfen. Ursächlich ist also tatsächlich, dass der benutzte Farbverwaltungsmechanismus nicht in natürlicher Weise mit den TEX-Gruppen synchronisiert ist.

Das hier dargestellte Problem lässt sich sehr einfach lösen, indem man im Innern der Box eine zusätzliche Gruppe so einfügt, dass das Gruppenende früh genug auftritt, um das Farbwiederherstellungs-Whatsit noch *in* der Box abzulegen. Die LATEX-Makros zur Behandlung von Boxen (`\sbox`, `\savebox`, `lrbox`) ergreifen diese Vorsichtsmaßnahme von sich aus; das Problem ist also gelöst, wenn man nur diese Makros nutzt. Wenn man auch auf Primitiv-Befehle zurückgreifen will, kann man den Inhalt der Boxen mit `\color@begingroup` und `\color@endgroup` umgeben: Diese Makros sind äquivalent zu `\begingroup` und `\endgroup`, wenn `color` geladen ist; andernfalls sind sie gleichbedeutend mit `\relax`, was einen etwas effizienteren Code erzeugt.

Die Effizienz des Codes allein ist jedoch nicht ausschlaggebend. Nach Meinung des Autors liegt der wirklich stichhaltige Grund, diese LATEX-Makros (an Stelle von Primitiven oder einem einfachen Paar geschweifter Klammern) zu benutzen darin, dass ihr Name klar auf die Nützlichkeit dieser zusätzlichen Gruppe hinweist, die einem zerstreuten (oder für dieses Problem nicht sensibilisierten) Leser andernfalls überflüssig erscheinen könnte. Zusammengefasst ist das Problem gelöst, wenn man die dritte Zeile wahlweise durch `\sbox0{\color{green} Box}%` oder durch

```
\setbox0\hbox{\color@begingroup\color{green} Box\color@endgroup}%
```

ersetzt. Befassen wir uns nun mit dem zweiten Punkt: Farbe als Eigenschaft einer Zone und nicht der einzelnen Knoten. Das kann man gut an folgendem Beispiel sehen:

```
\setbox0\hbox{schwarz}
\textcolor{gray}{Kein \usebox0\ nirgends.}
```

Das Wort »schwarz« wird grau gedruckt. Es erbt die Farbe seiner Umgebung. Das kann man als wünschenswert oder im Gegenteil als Problem betrachten. Was man auf jeden Fall sagen kann (ohne Stellung zu beziehen) ist, dass das Verhalten der Farben in diesem Punkt von dem der Zeichensätze abweicht. Es gibt jedoch mehrere Umstände, wo dies objektiv ein Problem darstellt: Stellen wir uns zum Beispiel eine `float`-Umgebung vor, die an einer Stelle eingefügt wird, an der sich die aktuelle Farbe von der Standardfarbe unterscheidet und deren Farbe so verändert werden würde.

Auch hier stellt LATEX eine robuste Lösung bereit, wenn man nur daran denkt, sie zu benutzen (oder sich auf die Makros des LATEX-Kerns zu verlassen, die von ihr

Gebrauch machen): Es handelt sich darum, den Teil des Dokuments, dessen Farbe nicht mehr verändert werden soll, in `\color@setgroup` und `\color@endgroup` einzuschließen. Das zweite dieser Makros kennen wir bereits. Das erste wird (wenn `color.sty` geladen ist) definiert als `\begingroup\set@color`. Es erzeugt so eine Farbzone, die die aktuelle Farbe benutzt, welche in `\current@color` abgespeichert ist, das immer den richtigen Wert enthält, wie wir in dem vorangegangenen Beispiel gesehen haben.

Man erhält so in unserem Beispiel ein Verhalten, das mehr dem der Zeichensätze entspricht, indem man seine erste Zeile durch `\sbox0{schwarz}` oder durch

```
\setbox0\hbox{\color@setgroup schwarz\color@endgroup}
```

ersetzt. Wir merken an, dass dies zugleich das erste angesprochene Problem löst. Es ist also nicht sinnvoll, beide Lösungen gleichzeitig zu benutzen: Man verwendet entweder `\color@begingroup` (wenn man die Farbe nicht »einfrieren« will) oder `\color@setgroup`, aber nicht beide gleichzeitig.

Wenden wir uns nun dem dritten Punkt zu: den Whatsit-Knoten. Das Problem besteht darin, dass diese zusätzliche Knoten in der aktuellen Liste sind und damit möglicherweise nicht unsichtbar bleiben. Mindestens zwei Arten von Problemen können sich stellen: Primitivbefehle, die das letzte Glied der Liste abfragen oder bearbeiten (`\unskip`, `\lastskip`, `\lastkern` usw.), könnten aus Versehen an Stelle des Leerzeichens, Leerraums oder etwas Anderem, das sie lesen wollten, den Whatsit-Knoten »sehen«. In einem LaTeX-Kontext könnte ein relativ natürliches Beispiel folgendermaßen aussehen:

```
\begin{center} \color{gray}
  Wichtiger Hinweis!
\end{center}
\addvspace{1ex}\hrulefill\par
```

Erinnern wir uns, dass `\addvspace` versucht, den vorangegangenen vertikalen Abstand zu berücksichtigen. Es vergleicht den geforderten Abstand mit dem vorigen und setzt den Abstand so, dass der größere der beiden übrigbleibt. Vergleichen wir das Ergebnis des Ausschnitts mit und ohne Farbwechsel:

Wichtiger Hinweis!

Wichtiger Hinweis!

Durch den Whatsit-Knoten wird der vorige Abstand vor `\addvspace` verborgen, so dass das Makro seinen Abstand immer zusätzlich setzt.

Das andere Problem tritt zum Beispiel auf, wenn man eine vertikale Box erstellen und die Farbe ihres gesamten Inhalts ändern möchte. Das folgende Beispiel ist vollkommen künstlich, veranschaulicht aber gut, worum es geht:

```
x\parbox[t]{1ex}{\color{gray} x}x
x\parbox[t]{1ex}{\textcolor{gray}{x}}x
```

Das Ergebnis spricht für sich: »x x xxx«. Ziel ist es, die Boxen an ihrer ersten Zeile aneinander auszurichten (interne Benutzung von `\vtop`). Im ersten Fall erzeugt `\color` ein Farbänderungs-Whatsit, das das erste Element der laufenden vertikalen Liste wird (Whatsit-Knoten können im vertikalem Modus ebenso wie im horizontalem Modus auftreten). Das »x« wird also quasi die zweite Zeile, und die Zeilenausrichtung findet an seiner Oberseite (der Unterseite der vorangehenden Zeile, die nur den Whatsit enthält) statt. Im zweiten Fall führt `\textcolor`, das nur innerhalb eines Absatzes verwendet werden soll, als Erstes ein `\leavevmode` aus, so dass das Farbänderungs-Whatsit Teil der horizontalen Liste wird, die den Absatz ausmacht. Sobald dieser gebildet worden ist, ist seine erste Zeile genau diejenige, die das »x« mit dem vorangehenden Whatsit enthält, und die erwartete Zeilenausrichtung ist erreicht.

In beiden Beispielen, die die von der Anwesenheit von Whatsit-Knoten verursachten Schwierigkeiten darstellen, sieht man, dass es möglich ist, dies von Hand zu korrigieren. Im Übrigen sind diese Probleme den Experten sehr wohl bekannt und dokumentiert worden. Dennoch kann das beobachtete Verhalten einem nicht vorgewarnten Benutzer überraschend und geheimnisvoll erscheinen. Außerdem scheint keine allgemeine Lösung zur Verfügung zu stehen. (Nur die Probleme mit `\unskip` und `\last`⟨…⟩ können vielleicht in LuaTEX umschifft werden, indem man den Gebrauch dieser Befehle durch ein Durchlaufen der Knotenlisten in Lua ersetzt.)

Insgesamt haben wir gesehen, dass die gängige Implementierung der Farben eine Reihe von Problemen bereitet, von denen einige mit einer gewissen Disziplin beim Programmieren gelöst werden können, wohingegen andere komplexer erscheinen. Wir werden jetzt die Implementierung der Farben mit Hilfe von Attributen, die die beschriebenen Probleme vermeidet und eine natürlichere Lösung zur Verfügung stellt, untersuchen.

Anwendung der Attribute

In diesem Abschnitt werden wir ausschließlich in der LATEX-Welt bleiben und ein Paket studieren, das die Farben auf Basis der Attribute neu implementiert: `luacolor`. Wir zeigen den Code dieses Pakets in seiner ganzen Komplexität; einige Aspekte lassen wir aber für das bessere Verständnis weg (insbesondere die Aufteilung in Funktionen und ihre Reihenfolge, Maßnahmen zur Kompatibilität, die Besonderheiten für den DVI-Modus und eine Optimierung der `\leaders`-Boxen). Jedes Weglassen wird für den neugierigen Leser, der sich unter [9] den

vollständigen Code anschauen kann, angegeben. Die Beschreibung hier basiert auf der Version 1.6 des Pakets `luacolor`.

Das Grundprinzip: Die aktuelle Farbe wird in einem Attribut gespeichert. Dieses wird automatisch von LuaTEX an die Knoten angehängt und folgt den üblichen Gültigkeitsregeln von TEX. Später werden diese Werte in den Attributen beim Durchlaufen der Knotenliste, aus der die Seite besteht, analysiert, und für jede Farbänderung wird ein Whatsit-Knoten eingefügt, damit der Farbwechsel in der Ausgabedatei (PDF oder DVI) sichtbar wird. Diese Whatsit-Knoten mit den Farbangaben werden dann von den Ausgabetreibern benötigt. Dieser Schritt sollte möglichst spät erfolgen, damit die Farbangaben durch Attribute lange erhalten bleiben, da dies eher dem natürlichen Verhalten von TEX entspricht.

Zwei problematische Aspekte bei den Farben bleiben:

- Die Notwendigkeit, mehrere Farbmodelle aus der Anwendersicht zu verwalten und anschließend in ein anderes Farbmodell zu konvertieren.
- Die Vielfalt der Ausgabeformate. Man könnte meinen, dass das einzige angemessene Ausgabeformat PDF ist, da DVI einige der interessanten Eigenschaften von LuaTEX nicht unterstützt (OpenType, Unicode). Es gibt aber keinen Grund, warum nicht in Zukunft auch andere Ausgabeformate zur Verfügung stehen, wie etwa SVG oder ein möglicher PDF-Nachfolger.

Die Lösungen für diese Probleme, die das Paket `color` bietet, bleiben weiterhin gültig. Insbesondere lädt `luacolor` das Paket `color` und erbt die Mechanismen der Treiberauswahl, seine allgemeinen Makros und die spezifischen Makros des ausgewählten Treibers. Aus Benutzersicht lädt man einfach das Paket `luacolor` anstelle von `color`.

Lassen Sie uns nun ins Detail gehen und mit dem TEX-Teil von `luacolor` anfangen.

TEX-Makros

Wir übergehen die Identifikation des Pakets, die Normalisierung der Catcodes, das Laden von Hilfspaketen wie z. B. `luatex.sty` oder `luatexbase-attr`, das in Abschnitt »Wichtige Erweiterungen« beschrieben wurde, sowie das Laden des Pakets `color`, falls es noch nicht geladen wurde. Wir überspringen den Test, ob auch wirklich LuaTEX benutzt wird, sowie die Behandlung des trivialen Falls, dass `color` mit der Option `monochrome` geladen wurde.

Die erste Aktion, die uns hier interessiert, ist das Laden des Lua-Codes:

```
\directlua{require("oberdiek.luacolor")}
```

`require` ist eine Standard-Funktion von Lua und lädt ein Modul. Diese Funktion wird durch LuaTEX verändert [8, § 3.2], so dass `require` im normalen TEX-Verzeichnisbaum ein Modul nachlädt. Die Punkte im Namen des Moduls werden in Verzeichnis-Separatoren bei der Suche umgewandelt.

Dem Modul selbst wird der Name des Autors vorangestellt: Damit können viele Pakete vom Autor Heiko Oberdiek in einem Verzeichnis gruppiert werden. Vor allem dient es als »Namensraum« (einer Lua-Tabelle). Dadurch wird vermieden, dass ein Konflikt bei Paketen, die denselben Namen verwenden, entsteht. Dieselbe Idee wird auch im LATEX-Teil des Quellcodes fortgeführt: Hier beginnen alle internen Befehle mit `\LuaCol@`, um das Risiko von Namenskonflikten mit anderen Paketen zu reduzieren.

Es wird nun ein Attribut-Register reserviert, um die aktuelle Farbe zu speichern (wir werden später sehen, dass ein einzelnes Attribut dazu ausreicht), und die Registernummer wird dem Lua-Code übergeben.

```
\newattribute\LuaCol@Attribute
\directlua{
  oberdiek.luacolor.setattribute(\number\allocationnumber)
}
```

Nach der Zuweisung enthält `\allocationnumber` die Nummer des Attribut-Registers, das ihm zugeordnet wurde, und wird mit der Lua-Funktion `setattribute` gespeichert.

Nun werden die Makros von `color` verändert, damit diese das neue Attribut verwenden. Dazu reicht es, `\set@color` und `\reset@color` umzudefinieren:

```
\protected\def\set@color{%
  \setattribute\LuaCol@Attribute{%
    \LuaCol@directlua{%
      oberdiek.luacolor.get("\luatexluaescapestring{\current@color}")%
    }%
  }%
}
\def\reset@color{}
```

Das Makro `\reset@color` (und dessen Benutzung mit `\aftergroup` in `\set@color`) wird nicht mehr benötigt, da die Werte der Attribute TEXs Gruppenschachtelung folgen. `\set@color` hat nur eine Aufgabe: Es muss dem Attribut den Wert der aktuellen Farbe geben, die `\current@color` enthält. Die Konvertierungsfunktion `get`, um aus einer Farbangabe einen Zahlenwert für das Attribut zu machen, wird später erklärt.

Damit haben wir, gespeichert an jeden Knoten, eine Repräsentation seiner Farbe als Zahl, die wiederum eine Zeichenkette repräsentiert, die im Ausgabeformat benutzt werden kann. Nun muss eine Lua-Funktion aufgerufen werden, die diese

Information benutzt. Das folgende Makro macht genau dies für eine Box, die durch ihre Nummer identifiziert wird.

```
\def\luacolorProcessBox#1{%
  \directlua{oberdiek.luacolor.process(\number#1)}%
}
```

Nun müssen wir dafür sorgen, dass das Makro (bzw. die zugrunde liegende Lua-Funktion) für die Box aufgerufen wird, die die auszugebende Seite beinhaltet. Also wird der Code mit der Seitenausgabe von TEX (`\shipout`) verbunden. Dies wird mithilfe des Pakets `atbegshi` erledigt, das für einen solchen Einsatzzweck gedacht ist. LuaTEX stellt keine solche Schnittstelle als Callback zur Verfügung.

```
\RequirePackage{atbegshi}[2007/09/09]
\AtBeginShipout{%
  \luacolorProcessBox\AtBeginShipoutBox
}
```

Wir sehen, dass die meisten TEX-Makros einfache Wrapper um Lua-Funktionen sind, die den größten Teil der Arbeit verrichten. Bevor wir die Details dieser Funktionen ansehen, ist es wichtig zu wissen, dass bei der Übergabe von Argumenten von TEX an Lua gewisse Vorsichtsmaßnahmen notwendig sind: Zeichenketten werden mit `\luatexluaescapestring` »gesichert« und Zahlen mit `\number` in explizite Zahlenangaben umgewandelt. Solche Vorsichtsmaßnahmen sollten systematisch benutzt werden.

Der Lua-Code

Er beginnt, wie üblich, mit dem Aufruf der Standard-Funktion `module`:

```
module("oberdiek.luacolor", package.seeall)
```

Bevor Missverständnisse auftreten: Diese Funktion ist kein Pendant zum LATEX-Makro `\ProvidesPackage`. Seine Aufgabe ist es nicht, zu überprüfen, dass das Modul zum passenden `require`-Aufruf passt (die beiden Werte von `require` und `module` müssen nicht zusammen passen, obwohl das gängige Praxis ist). Vielmehr wird durch das erste Argument von `module` ein aktueller Namensraum definiert (in Lua auch »environment« genannt). Allen Variablen (Lua-Funktionen fallen auch darunter) wird bis zum Ende der Datei ein »`oberdiek.luacolor`« vorangestellt. Das zweite Argument `package.seeall` bewirkt, dass alle globalen Variablen (und Funktionen) in diesem Modul sichtbar und damit benutzbar sind. Ein Beispiel dafür ist die Tabelle `tex`.

Zunächst wird eine lokale Variable reserviert, die die Nummer des zu verwendenden Attributes speichert. Die Funktion `setattribute` speichert diese Nummer.

```
local attribute
function setattribute(attr)
  attribute = attr
end
```

Diese Funktion ist in dieser Datei unter dem genannten Namen sichtbar; außerhalb des Moduls muss der vollständige Name `oberdiek.luacolor.setattribute` angegeben werden. Die Variable `attribute` ist nur für diese Datei sichtbar und damit nicht von anderen Modulen zugänglich. Eine gute Praxis ist es, alle Variablen und Funktionen mit der Sichtbarkeit »lokal« zu definieren und nur die Funktionen, die von außen zugänglich sein sollen, »global« zu lassen.

Nun zu einem schwierigen Punkt: Der Wert eines Attributes kann nur eine ganze Zahl sein,[8] aber der Wert der aktuellen Farbe (in `\current@color` gemerkt) ist eine Zeichenkette. Es muss daher ein Weg gefunden werden, die Zeichenkette mit einer Zahl zu verbinden, damit diese später wieder in eine Zeichenkette umgewandelt werden kann. Das ist die Aufgabe der Funktion `luacolor.get`, die hier gezeigt wird.

```
local map = { n = 0 }
function get(color)
  local n = map[color]
  if not n then
    n = map.n + 1
    map.n = n
    map[n] = color
    map[color] = n
  end
  tex.write("" .. n)
end
```

Das Grundprinzip ist einfach: In einer Tabelle werden alle Namen der Farben in der Reihenfolge ihrer ersten Verwendung nummeriert gespeichert. Die Umsetzung ist elegant und erlaubt uns, einige Eigenschaften von Lua-Tabellen zu zeigen. Die Tabelle ist eine der grundlegenden Datenstrukturen in Lua. Sie dient sowohl als Feld (engl. Array), um Werte anhand eines numerischen Index zu speichern, als auch als Wörterbuch mit beliebigen Schlüsseln. Hier werden beide Eigenschaften gleichzeitig benutzt. Dadurch kann die Tabelle `map` sowohl den Farbwert als Zahl als auch zu einer Zahl den dazugehörenden Farbwert auffinden. Ein besonderer Eintrag `n` speichert die Zahl der Farbwerte bzw. der verwendeten Nummern. Zum Beispiel nehmen wir an, dass die erste benutzte Farbe Schwarz ist und die zweite Rot und dass wir PDF als Ausgabeformat haben. Direkt nach der ersten Benutzung von Rot hat die Tabelle `map` die folgende Struktur:

[8] Eine positive Zahl, wenn die Kompatibilität mit älteren LuaTEX-Versionen wichtig ist.

```
{
  ['n'] = 2,
  [1] = '0 g 0 G',
  ['0 g 0 G'] = 1,
  [2] = '1 0 0 rg 1 0 0 RG',
  ['1 0 0 rg 1 0 0 RG'] = 2,
}
```

Zum besseren Verständnis der Funktion und der zugrunde liegenden Tabelle: Denken Sie daran, dass Lua zwei gleichwertige Schreibweisen für Zeichenketten-Schlüssel hat: Die weiter oben stehende erste Deklaration `n = ⟨Wert⟩` ist äquivalent zu `['n'] = ⟨Wert⟩`. Um auf diesen Wert wieder zuzugreifen, ist `map.n` äquivalent zu `map['n']`. Im Gegensatz dazu ist in `map[n]` das `n` der Name einer Variablen, deren Wert hier als numerischer Index benutzt wird, also beispielsweise `map[2]`.

Dieser Mechanismus zum (virtuellen) Speichern von Zeichenketten in Attributen kann immer verwendet werden. Die einzige Einschränkung ist, dass nicht mehr als 2147483647 verschiedene Werte in einem Dokument verwendet werden dürfen.

Die wichtigste Funktion ist die, die eine Box durchläuft und die Farb-Attribute untersucht, um Farbwechsel-Whatsits an den geeigneten Stellen einzufügen (`traverse`). Da eine Box auch selbst Boxen enthalten kann, ruft diese Funktion sich selbst rekursiv auf. Zum Einstieg benötigen wir eine Funktion, die zuerst aus der übergebenen Boxnummer den Boxinhalt (Box-Knoten) holt und diesen dann an die rekursive Funktion `traverse` übergibt.

```
function process(box)
  local color = ""
  local list = tex.getbox(box)
  traverse(list, color)
end
```

Die Funktion `traverse` merkt sich zu jedem Zeitpunkt die Farbe des Vorgängerknotens, um vor dem aktuellen Knoten ein Whatsit einfügen zu können, wenn sich die Farbe im Vergleich zum Vorgängerknoten geändert hat. Vor der ersten Benutzung beginnt die Funktion mit einer leeren Zeichenkette, die keiner Farbe entspricht. Damit erzwingt der erste farbrelevante Knoten einer Seite das Setzen der Farbe.[9]

Die Funktion `traverse` interessiert sich nicht für alle Knoten, sondern nur für die folgenden, die für die Farben relevant sind:

[9] Falls die erste Farbe Schwarz ist, könnte man theoretisch diese erste Farbsetzung im PDF-Format weglassen.

- Boxen oder `leaders`, die Boxen enthalten, insbesondere, weil diese wiederum farblich relevante Knoten enthalten können.
- Andere sichtbare Knoten (im Gegensatz z. B. zu Leerraum oder Leerzeichen[10]).

Jetzt werden erst einmal die Konstanten, die diese Nodetypen identifizieren, und zwei Funktionen deklariert. Dann befassen wir uns direkt mit dem Hauptteil der Funktion `traverse`:

```
local LIST, LIST_LEADERS, COLOR = 1, 2, 3
local get_type, build_whatsit
function traverse(list, color)
  if not list then
    return color
  end
  if get_type(list) ~= LIST then
    texio.write_nl(
      "!!! Error: Wrong list type: " .. node.type(list.id)
    )
    return color
  end
```

Die Funktion beginnt mit der Behandlung von Sonderfällen: Bei einer leeren Box sind wir bereits fertig. Wenn die Funktion mit einem anderen Argument als eine Box aufgerufen wird, wird eine Fehlermeldung ausgegeben.

Da wir jetzt sichergestellt haben, dass wir eine gültige Box vorfinden, können wir ihren Inhalt durchlaufen und unterscheiden die Knotentypen:

```
  local head = list.head
  for n in node.traverse(head) do
    local t = get_type(n)
    if t == LIST then
      color = traverse(n, color)
```

Wenn wir es mit einem Knoten vom Typ »Liste« zu tun haben, also wieder einer Box, dann verarbeiten wir diese rekursiv. Dabei übergeben wir dem Aufruf die Farbe des letzten sichtbaren Knotens und erhalten die Farbe des letzten sichtbaren Knotens der Box, damit wir wieder die Werte vergleichen können.

Einen Sonderfall bilden die Boxen, die man bei `\leaders` (bzw. `\cleaders`[11] oder `\xleaders`) verwendet. Je nach dem Platz, der durch diese `\leaders`-Boxen abgedeckt werden soll, wird diese Box von TEX überhaupt nicht, einmal oder gar mehrmals hintereinander gesetzt. Am Anfang einer `\leaders`-Box wissen wir an dieser Stelle nicht, ob der Vorgängerknoten vom vorigen Text stammt oder ob wir uns bereits in einer wiederholten Box befinden, der Vorgängerknoten somit am

[10] Leerzeichen sind in TEX als Leerraum implementiert.
[11] Dieses wird intern von LATEXs `\dotfill` verwendet.

Ende der Box liegt. Als Vorsichtsmaßnahme setzen wir daher an dieser unklaren Stelle die aktuelle Farbe auf die leere Zeichenkette, um die Ausgabe der Farb-Whatsits zu erzwingen.[12]

```
    elseif t == LIST_LEADERS then
      traverse(n.leader, '')
      color = ''
```

In die gleiche Richtung geht die Vorsichtsmaßnahme direkt nach den `\leaders`. Man kennt die Farbe des letzten sichtbaren Elements nicht, weil die `\leaders`-Box möglicherweise aus Platzmangel gar nicht eingefügt wurde, und wir haben keine (einfache) Möglichkeit, das herauszufinden. Daher wird die nicht existierende Farbe (die leere Zeichenkette) als letzte sichtbare Farbe angenommen.

Der Fall der sichtbaren Knoten liegt gänzlich anders. Man kann den Farbwert aus dem Attribut holen und ihn mit dem vorherigen Wert vergleichen. Wenn er unterschiedlich ist, wird die aktuelle Farbe auf den neuen Wert gesetzt (um ihn mit den nächsten Knoten zu vergleichen) und es wird vor dem aktuellen Knoten ein Whatsit-Knoten für den Farbwechsel eingefügt. Dieser wird mithilfe einer Funktion erzeugt, die weiter unten vorgestellt wird.

```
    elseif t == COLOR then
      local v = node.has_attribute(n, attribute)
      if v then
        local newColor = map[v]
        if newColor ~= color then
          color = newColor
          head = node.insert_before(head, n, build_whatsit(color))
        end
      end
```

Andere Knotentypen werden nicht behandelt, daher enden hier die Fallunterscheidungen bezüglich des Knotentyps, und der Durchlauf des Boxinhalts wird beendet:

```
    end
  end
  list.head = head
  return color
end
```

Es ist wichtig, vor dem Verlassen der Funktion (und der Rückgabe der Farbe des letzten sichtbaren Knotens) den Wert von `list.head` (der Zeiger auf den ersten Knoten des Inhalts der Box) zu aktualisieren. Falls ein Farbwechsel gleich am

[12] In Wirklichkeit ist der Code von `luacolor` ein wenig komplexer und bestimmt in einem ersten Durchlauf, ob sich die Farbe in einer `\leaders`-Box überhaupt geändert hat, um dann nur die benötigten Whatsits zu setzen. Der hier vorgestellte Code ist eine vereinfachte Fassung, aber voll funktionstüchtig. Nur kann er manchmal überflüssige Whatsits einfügen.

Anfang der Box stattgefunden hat, wurde ein Whatsit-Knoten vor dem ersten Element eingefügt. Das kann beim Einfügen von Knoten in Boxen passieren und daher sollte man immer vorsichtig sein.

Sehen wir uns nun die Klassifizierung der Knoten in die oben genannten Kategorien an: Boxen, `leaders`, die Boxen benutzen, und andere sichtbare Knoten. Dafür benutzen wir eine Tabelle für die Rohdaten und eine Funktion, um darauf zuzugreifen.

```
local RULE = node.id("rule")
local node_types = {
  [node.id("hlist")] = LIST,
  [node.id("vlist")] = LIST,
  [node.id("rule")] = COLOR,
  [node.id("glyph")] = COLOR,
  [node.id("disc")] = COLOR,
  [node.id("whatsit")] = {
     [3] = COLOR, -- special
     [8] = COLOR, -- pdf_literal
    [14] = COLOR, -- pdf_refximage
  },
  [node.id("glue")] = function(n)
     if n.subtype >= 100 then -- leaders
       if n.leader.id == RULE then
         return COLOR
       else
         return LIST_LEADERS
       end
     end
  end,
}
get_type = function (n)
  local ret = node_types[n.id]
  if type(ret) == 'table' then
    ret = ret[n.subtype]
  end
  if type(ret) == 'function' then
    ret = ret(n)
  end
  return ret
end
```

Zuerst versuchen wir die Knoten anhand ihres `id`-Feldes einzuteilen. (Dieses Feld enthält eine Zahl, die ihren Typ repräsentiert.) Mithilfe der Funktion `node.id` kann man sie leicht mit menschenlesbaren Werten – z. B. `'hlist'` – vergleichen. Einige Typen sind zu allgemein und benötigen eine Untertabelle, um sie anhand ihrer Untertypen zu klassifizieren. Zum Beispiel sind die Whatsit-Knoten oft unsichtbar (wenn etwa mit `\write` etwas in eine Datei geschrieben wird), manche aber sind sichtbar oder können sichtbar sein. Das ist der Fall bei `\special` und

`\pdfliteral`, die beliebiges Material erzeugen können. Im Gegensatz zu den Knotentypen wird hier der numerische Wert direkt verwendet (dieser könnte aber mit der Funktion `node.subtype` ebenso ermittelt werden). Diese Werte lassen sich im LuaTEX-Handbuch nachschlagen.

Einige Knoten sind so komplex, dass Typ und Untertyp nicht ausreichen, um zu entscheiden, in welche Kategorie er fällt. Also benötigen wir eine spezielle Funktion (hier sehen Sie nebenbei die einfache Benutzung von anonymen Funktionen in Lua). Das ist der Fall bei Glue-Knoten, die normalerweise unsichtbar sind, mit Ausnahme bei `\leaders`, die anhand ihrer Untertypen erkannt werden. Aber es gibt verschiedene Arten von `\leaders`: diejenigen, die Linien verwenden, sind normal sichtbar und die anderen, die Boxen verwenden, kann man wie oben beschrieben mithilfe von Rekursion untersuchen.

An dieser Stelle mag sich der Leser fragen, woher dieser Inhalt der Tabelle `node_types` kommt. Es gibt hierauf keine einfache Antwort, die beste Annäherung an die Wahrheit besteht aus einer Mischung von Kenntnissen von TEX, der Lektüre des LuaTEX-Handbuchs und von Experimenten. Zum Beispiel hat der Autor dieses Artikels bei einem Beispiel mit `luacolor` entdeckt, dass bestimmte Glue-Knoten sichtbar waren. Nach Lektüre des Handbuchs und erneutem Lesen von *The TEXbook* [7] und *TEX by topic* [3] erkannte er, dass man die Boxen bestimmter `\leaders` rekursiv untersuchen muss und hat daraufhin eine Korrektur an den Autor des Pakets `luacolor` gesendet, unterstützt durch eine für seine psychische Gesundheit wichtige Testdatei.[13]

Es muss nur noch eine Funktion definiert werden, die einen Whatsit-Knoten für den Farbwechsel erzeugt. Diese Funktion ist abhängig vom Ausgabeformat.

```
local WHATSIT = node.id("whatsit")
if tex.pdfoutput > 0 then
  local PDFLITERAL = 8
  local mode = 2 -- direct
  function build_whatsit(color)
    w = node.new(WHATSIT, PDFLITERAL)
    w.mode = mode
    w.data = color
    return w
  end
end
```

Wie üblich definieren wir einige Konstanten anstatt direkt mit konkreten Zahlenwerten zu arbeiten. (Die Informatiker nennen sie »magische Zahlen«, weil beim Lesen des Codes der Eindruck entsteht, dass ihr Wert durch Magie bestimmt wurde.) Der Wert des Knotentyps Whatsit wird wie vorher mit `node.id`

[13] Anm. vom Paketautor: Sehr gut! Zur Nachahmung wärmstens empfohlen.

bestimmt; der Wert des Subtyps `pdf_literal` mithilfe des Handbuchs[14] und der von `mode` mithilfe einer zukünftigen Version des Handbuchs (vermutlich der Version 0.70). Zum Schluss erzeugt die Funktion denselben Knoten wie `\pdfliteral direct {⟨<color>⟩}`. Das Schlüsselwort `direct` wird verwendet, damit der PDF-Code direkt eingefügt wird ohne Vorsichtsmaßnahmen in Bezug auf den Zusammenhang zu machen [4, § 7.2].

Es mag auf den ersten Blick überraschen, dass `\pdfliteral` anstelle von `\pdfcolorstack` benutzt wird. In der Tat ist die Benutzung eines Farb-Stapels (*stack*) nicht mehr nötig. Seine Rolle übernehmen die Gruppen von TEX, denen die Attributwerte folgen. Dieser Wechsel ist daher im Einklang mit dem Ziel, die Farben von TEX eng mit den Gruppen zu verbinden.

Um den ohnehin schon langen technischen Abschnitt nicht noch weiter in die Länge zu treiben, vernachlässigen wir den DVI-Modus. Wir erwähnen aber das elegante und clevere Prinzip: Bevor die Farbverwaltung umdefiniert wird, wird (von TEX) eine Box mit einem Farbwechsel erstellt, um sie anschließend mit Lua zu untersuchen und daraus die benötigten `\special`-Whatsits zu erstellen, die dasselbe wie die `\pdfliteral`-Whatsits machen (wie oben beschrieben).

Letztendlich kann `luacolor` einen Großteil der Farbverwaltung mithilfe von in Attributen gespeicherten Farben erledigen und die notwendigen Whatsit-Knoten, die wie bereits erwähnt auch Probleme bereiten, so spät wie möglich einfügen. Die Umsetzung ist sauber und robust (im Gegensatz zu *schmutzigen Tricks*, die vorher vorkamen). Sie ist aber technisch und erfordert Kenntnisse in einigen Details von TEX.

Schlussfolgerung

TEX besitzt einen leistungsfähigen Erweiterungsmechanismus, der größtenteils von pdfTEX verwendet wird: Whatsits. LuaTEX bietet einen neuen Erweiterungsmechanismus, der es erlaubt, jedem Knoten eine Zahl von logischen Charakteristiken zuzuweisen, der dem üblichen Gruppierungsverhalten von TEX folgt und auf den frei in Lua zugegriffen werden kann: die Attribute.

Wir hoffen, dass wir zeigen konnten, dass die Whatsits nicht perfekt auf den besonderen Fall der Farben zugeschnitten sind und die Attribute eine robustere und natürlichere Implementierung ermöglichen. Natürlich ist die Benutzung der Attribute nicht auf Farben beschränkt; man findet andere Beispiele ihrer Anwendung in [6].

[14] Anm. d. Übersetzer: Anstelle der Konstanten könnten auch die Zeichenketten `whatsit` und `pdf_literal` direkt benutzt werden.

Nebenbei sei angemerkt, dass die Handhabung der Sprache (d. h. welche Trennpattern für die Silbentrennung verwendet werden sollen) von pdfTEX zu LuaTEX eine ähnliche Verschiebung durchgemacht hat, wie die hier vorgestellte für die Farben: ausgehend von der Verwendung von Whatsit-Knoten hin zu individueller Speicherung in jedem Knoten. Allerdings werden wie bei den Schriften die Sprachangaben nicht mithilfe von Attributen verarbeitet, da diese direkt von dem Programm gebraucht werden, um etwa die Silbentrennungen zu ermitteln. Dagegen werden Attribute nur vom Programmierer verwendet.

Das ist übrigens das Schöne an den Attributen: Sie stellen einen flexiblen Erweiterungsmechanismus dar, den der Benutzer von LuaTEX verwenden kann. Im Gegensatz dazu können neue Whatsit-Typen nur durch Programmierer hinzugefügt werden, die das TEX-Programm entsprechend erweitern (wie pdfTEX im Verhältnis zu TEX3). Um den Whatsits Gerechtigkeit widerfahren zu lassen, hat LuaTEX einen Whatsit-Typ für den Anwender definiert: Die `user_defined`-Whatsits bieten eine ähnliche Flexibilität wie die Attribute.

Die Moral von der Geschichte könnte lauten: mehr Auswahl und Erweiterbarkeit auf allen Ebenen. Wer wird sich da beschweren?

Literatur

[1] David P. Carlisle: *The color package*; 2005; `CTAN:macros/latex/required/graphics/`.

[2] David P. Carlisle und The LATEX3 Project: *Packages in the 'graphics' bundle*; 2005; `CTAN:macros/latex/required/graphics/grfguide.pdf`.

[3] Victor Eijkhout: *TEX by Topic*; Addison-Wesley; 1992; `http://eijkhout.net/texbytopic/texbytopic.html`.

[4] Hàn Thế Thành, Sebastian Rahtz et al.: *The pdfTEX user manual*; 2007; `CTAN:systems/pdftex/manual/pdftex-a.pdf`.

[5] Roberto Ierusalimschy: *Programming in Lua*; Lua.org; 1. Aufl.; Dezember 2003; `http://lua.org/pil/`.

[6] Paul Isambert: *Three things you can do with LuaTEX that would be extremely painful otherwise*; TUGboat; 31, S. 184–190; 2010; `http://www.tug.org//Contents/contents31-3.html`.

[7] Donald E. Knuth: *The TEXbook*; Addison-Wesley; 1984.

[8] LuaTEX Development Team: *LuaTEX Reference*; Version 0.65.0; 2010; `CTAN:http://luatex.org/svn/tags/beta-0.65.0/manual/luatexref-t.pdf`.

[9] Heiko Oberdiek: *The luacolor package*; Version 1.6; 2011; `CTAN:macros/latex/contrib/oberdiek/luacolor.pdf`.

[10] Manuel Pégourié-Gonnard: *Un guide pour LuaLaTeX*; *Cahiers Gutenberg*; 54-55, S. 13–35; 2010; http://cahiers.gutenberg.eu.org/cg-bin/fitem?id=CG_2010___54-55_13_0.

[11] Sebastian Rahtz und David P. Carlisle: *Graphics drivers for LaTeX 2ε*; 2009; CTAN:macros/latex/required/graphics/.

Das Paket chickenize – Spaß mit Node-Manipulationen in LuaTEX

Arno Trautmann

Dieser Artikel ist ein Anschluss an den Vortrag des Autors [5] bei der DANTE-Herbsttagung 2011. Anhand einfacher Beispiele wird erläutert, wie mit LuaTEX der Inhalt eines Dokuments beliebig beeinflusst und geändert werden kann. Dabei steht das spielerische Lernen von LuaTEX-Interna im Vordergrund.

Warum LuaTEX?

Für den normalen Nutzer werden die Vorteile von LuaTEX meist auf zwei Themen beschränkt: UTF-8-Unterstützung und das Verwenden beliebiger Systemschriften (OpenType/TrueType). Beides bietet auch XeTEX, aber ohne Font-Expansion. Zur Entwicklungsgeschichte der verschiedenen TEX-Varianten siehe [6]. Auch einfache Anwendungen der integrierten Sprache Lua können für den Endnutzer interessant sein, wie Markus Kohm und Alexander Willand am Beispiel von Tabellen gezeigt haben. [4]

Eine der wichtigsten Neuerungen in LuaTEX bleibt aber den meisten Nutzern verborgen: Die Möglichkeit, den Arbeitsablauf von TEX an verschiedenen Stellen beliebig zu beeinflussen. Auf diesen sogenannten Callbacks basiert die eigentliche Stärke von LuaTEX.

Im Folgenden wird anhand zweier spezieller Callbacks, des pre_linebreak_filter und des post_linebreak_filter, die Funktionsweise und die Verwendung dieser Konstrukte demonstriert. Wo Codebeispiele aus Platzgründen gekürzt werden müssen, wird auf das unten vorgestellte Paket chickenize verwiesen.

Zum Verständnis der Callbacks muss allerdings zuerst ein weiterer wichtiger Begriff in LuaTEX erläutert werden, nämlich der Node-Begriff.

Nodes

In LuaTEX werden alle Objekte als verschiedene Typen von Nodes dargestellt. Im Folgenden werden u. a. Nodes vom Typ `glyph` verwendet, also Objekte, die einen Buchstaben darstellen. Weiterhin gibt es horizontale Boxen (`hlist`), dehnbare Abstände (`glue`), außerdem Linien (`rules`) und den speziellen Typ `whatsit`. Diese Typen entsprechen den TEX-Konstrukten mit den gleichen Namen.

Im Unterschied zur Arbeit auf der TEX-Seite können die Nodes auf der Lua-Seite manipuliert werden: Nodes können erzeugt, geändert, verschoben oder gelöscht werden, *nachdem* die Eingabe gelesen wurde.

Im normalen Arbeitsablauf liest LuaTEX eine Eingabezeile ein und wandelt die Zeichen in entsprechende Nodes um. Dabei entsteht meist eine lange Liste aus `glyph`- und `glue`-Node. Zur Vereinfachung wählen wir eine vereinfachte Darstellung ohne Trennstellen, Kerning usw.

Diese Liste wird dann vom Umbruchalgorithmus in `hlist`-Nodes zusammengebaut und als eine vertikale Liste (`vlist`) ausgegeben. Das ganze kann man entweder über die TEX-Seite bedienen, wie man es gewöhnt ist, oder aber ganz von der Lua-Seite, wie von Patrick Gundlach im LuaTEX-Wiki [1] beschrieben. Die `hlist`- und `vlist`-Nodes sind Tabellen, deren Feld `.head` (in früheren LuaTEX-Versionen `.list`) das erste Element und damit den eigentlichen Inhalt darstellt. Weitere Inhalte werden über dessen Feld `.next` in einer Liste aufgebaut.

Genau an diesen Stellen der Verarbeitung (Umbruch der horizontalen Liste zu einer vertikalen Liste) setzen die beiden hier verwendeten Callbacks an, direkt vor und nach dem Umbruchalgorithmus.

Callbacks

Callbacks bieten die Möglichkeit, beliebigen Lua-Code an bestimmten Stellen im Arbeitsablauf auszuführen. Das kann beim Schriftladen sein (wie das Paket `luaotfload`, durch das OpenType-Schriften geladen werden), bei der Seitenausgabe oder eben vor und nach dem Absatzumbruch. Manche Callbacks haben bereits vordefinierten Code, den man ändern oder ersetzen kann; die beiden `linebreak_filter` allerdings sind normalerweise leer, weshalb sie sich hervorragend für Beispiele eignen.

Damit ein Callback-Code ausführt, muss dieser zunächst in eine Lua-Funktion geschrieben werden und die Funktion dann im Callback »registriert« werden. Das kann entweder mit der Funktion `callback.register` von LuaTEX gemacht werden oder besser mit dem Paket `luatexbase`. Die Umgebung `luacode` aus dem gleichnamigen Paket erlaubt die einfache Eingabe von Lua-Code und wird im Weiteren für alle Beispiele vorausgesetzt:

```
\begin{luacode}
luatexbase.add_to_callback(
  "pre_linebreak_filter", myfunction, "meine Funktion")
\end{luacode}
```

Die Funktion `myfunction` wird damit vor dem Umbruchalgorithmus aufgerufen. Die Benennung `"meine Funktion"` ist ein Label, mit dem später auf die Funktion im Callback zugegriffen werden kann.

Mit diesem Wissen kann man nun in der LuaTEX-Dokumentation (`texdoc luatex`) nachlesen, wie dieser Callback definiert ist und wie die Funktion auszusehen hat. Für den pre_linebreak_filter findet man die Beschreibung:

> This callback is called just before LuaTEX starts converting a list of nodes into a stack of `\hboxes`, after the addition of `\parfillskip`.

Das ist das oben erklärte Verhalten. Weiterhin findet man die nötige Form der Funktion:

```
function(<node> head, <string> groupcode)
 return true | false | <node> newhead
end
```

Es wird also als erstes Argument ein Node erwartet, und der Rückgabewert ist entweder `true`, `false` oder ein neuer Node. Das zweite, optionale Argument wird im Weiteren nicht verwendet.

Einfache Funktionen registrieren

Die einfachste Möglichkeit, den pre_linebreak_filter zu nutzen, besteht in einer Funktion, die als Argument die horizontale Liste erhält und diese unverändert wieder ausgibt:

```
donothing = function(head)
  return head
end
luatexbase.add_to_callback(
  "pre_linebreak_filter", donothing, "nichts")
```

Betrachten wir als nächstes etwas Interessanteres, nämlich eine Funktion, die alle Buchstaben der Eingabe durchgeht. In weiteren Schritten können diese dann manipuliert werden:

```
allglyphs = function(head)
  for i in node.traverse_id(37, head) do
    -- beliebige Manipulation der Nodes
  end
  return head
end
```

```
luatexbase.add_to_callback(
  "pre_linebreak_filter", allglyphs, "alle Zeichen")
```

Sehr nützlich ist dabei die Funktion `node.traverse_id`. Diese geht alle Nodes der Liste durch, die mit dem Knoten `head` beginnt, allerdings *nur* für die Elemente, die dem angegebenen Node-Typen entsprechen, hier `37` für `glyph`-Nodes. Das spart die Abfrage, ob es sich beim betrachteten Node um einen Glyph handelt. Der doppelte Strich `--` ist das Kommentarzeichen in Lua.

Für den post_linebreak_filter ist die gleiche Aufgabe etwas komplexer, da nicht nur eine horizontale Liste vorliegt, sondern eine vertikale Liste, die aus horizontalen Listen besteht. Um also einen Absatz *nach* dem Umbruch buchstabenweise durchzugehen, muss eine doppelte Schleife zuerst alle horizontalen Boxen (Node-Typ `0`) durchgehen, um dann die gleiche Schleife wie oben zu verwenden:

```
allglyphs_post = function(head)
  -- zuerst durch alle hlist-nodes gehen
  for line in node.traveres_id(0, head)
    -- dann durch alle Zeichen in der Liste
    for i in node.traverse_id(37, line.head) do
      -- beliebige Manipulation der Nodes
    end
  end
  return head
end
```

Zu beachten ist dabei, dass die innere Schleife als Argument für `traverse_id` nicht den Knoten `line` erhält, sondern das erste Element der horizontalen Liste, `line.head`. Die Zeilen zum Registrieren der Funktion werden im Weiteren nicht mehr erwähnt. Neben horizontalen Boxen kann in einer vertikalen Liste auch z. B. `glue` vorkommen, daher muss auch hier der Node-Typ für die Schleife eingeschränkt werden.

Erste Manipulationen

Was stellt man nun mit diesen Möglichkeiten an? Man kann sich eine einfache, aber sehr lehrreiche Aufgabe stellen:

»Ersetze überall den Buchstaben `E` durch die Zahl `3`.«

Die schrittweise Lösung zum Problem lautet:

- Gehe *nach* dem Umbruch alle Buchstaben durch,
- prüfe, ob der Buchstabe ein `E` ist,
- ersetze ihn durch eine `3`,
- übergebe die geänderte Liste als Ausgabe.

Jeder Knoten `i` vom Typ `glyph` hat das Feld `i.char`, das den Unicodepunkt des Zeichens beinhaltet. Es wird also jetzt geprüft, ob der Codepunkt einem `E` entspricht und dieser dann gegebenenfalls einfach geändert:

```
leet = function(head)
  for line in node.traverse_id(0,head) do
    for i in node.traverse_id(37, line.head) do
      if i.char == 101 then -- ist der Buchstabe ein E?
        i.char = 51         -- ersetze durch eine 3
      end
    end
  end
  return head               -- Rückgabe der geänderten Liste
end
```

Registriert man diese Funktion im post_linebreak_filter, so wird der Text dementsprechend geändert. Der pre_linebreak_filter kann hier nicht verwendet werden, da sonst die Trennmuster nicht mehr funktionieren. Nachteilig ist, dass die Breite der Zeichen unterschiedlich ist und daher die Zeilenlänge nicht mehr stimmt – das lässt sich aber einfach wieder korrigieren. Mit etwas mehr Aufwand kann man den Text dann auch ganz in 1337-Schrift übersetzen oder beliebige Ersetzungen durchführen. Bei einer 1337-Schrift, auch Leetspeak genannt, werden Buchstaben durch ähnlich aussehende Zahlen ersetzt.

Farbspielereien

Gehen wir zu komplexeren Fragestellungen: Beim Heidelberger TEX-Stammtisch hat Markus Kohm vor einiger Zeit von folgender Anfrage eines Nutzers erzählt:

> »Wie kann ich jeden Großbuchstaben im Dokument bunt machen?«

Unter pdfTEX ist eine stabile Lösung unmöglich – für LuaTEX hingegen stellt es eine schöne, einfache Übung dar. Das Vorgehen ist in diesem Fall:

- Gehe nach dem Umbruch alle Buchstaben durch,
- prüfe, ob es sich um einen Großbuchstaben handelt,
- falls ja, baue einen »Farbe-Einfügen-Knoten« vor dem Buchstaben ein,
- füge einen »Farbe-Aufheben-Knoten« hinter dem Buchstaben ein,
- gib die geänderte Liste als Ausgabe zurück.

Der erste Teil ist analog zum Vorgehen oben, und das Einfügen von Nodes ist dank der Funktionen `node.insert_before` und `node.insert_after` sehr einfach zu erreichen.

Komplexer hingegen ist das Erzeugen von Farbknoten. Diese sind `whatsit`-Nodes vom Untertyp `pdf_colorstack`. Diese Node-Typen haben drei Eigenschaften, die

gesetzt werden müssen: Erstens das Feld `.stack`, hier immer `0`. Weiterhin muss die Aktion im Feld `.cmd` gesetzt werden. Mit dem Wert `1` wird eine Farbe auf den Stapel gesetzt, mit `2` wird die letzte Farbe entfernt (Anwendung der `push`- und `pop`-Operation). Die jeweils oberste Farbe wird dann für den Buchstaben verwendet.

Es werden nun zwei Farb-Nodes erzeugt, und zwar global (also nicht innerhalb der Funktion selbst):

```
color_push = node.new(8,39)
color_pop  = node.new(8,39)
```

Die beiden Zahlen geben dabei an, dass es sich um `whatsit`-Nodes (`8`) vom Subtyp `pdf_colorstack` (`39`) handelt. Die weiteren Eigenschaften werden ebenfalls global gesetzt, für die Farbe wird im Feld `.data` ein String für den RGB-Code angegeben:

```
color_push.stack = 0       color_pop.stack = 0
color_push.cmd   = 1       color_pop.cmd   = 2
color_push.data = "0.9 0.1 0.2 rg" -- kräftiges rot
```

Damit ist alles zusammen, um jeden Großbuchstaben farbig zu markieren:

```
uppercasecolor = function (head)
  for line in node.traverse_id(0,head) do
    for upper in node.traverse_id(37,line.head) do
      if ((upper.char > 64) and (upper.char < 91)) then
-- lateinische Großbuchstaben haben einen
-- Unicodepunkt zwischen 65 und 90
        line.head = node.insert_before(
                        line.head, upper, node.copy(color_push))
        node.insert_after(line.head, upper, node.copy(color_pop))
      end
    end
  end
  return head
end
```

Fängt eine Zeile mit einem Großbuchstaben an, so wird der Farbknoten *vor* dem Element `.head` der Zeile eingefügt. Im weiteren Verlauf würde der Knoten dann einfach ignoriert werden, daher muss das Element `line.head` explizit neu gesetzt werden.

Die Farbe kann bei jedem Buchstaben variiert werden, indem `color_push.data` neu gesetzt wird, auch mit Zufallszahlen mittels `math.random()`. Der nächsten Fastnachtseinladung steht dann nichts mehr im Wege!

Grauwert anzeigen

Während die bisherigen Übungen eher als lehrreiche Spielereien zu sehen sind, kann die nächste Aufgabe Informationen über die Qualität des Textsatzes angeben:

„Stelle den Grauwert (das Maß der Dehnung oder Stauchung) jeder Zeile mit einem grauen Balken dar.“

Je stärker die Zeile gestaucht werden muss, desto dunkler soll dabei der Balken sein. Die Lösung dafür folgt einem Ansatz von Paul Isambert [3] und soll hier nur skizziert werden: Die wichtige Eigenschaft, die ausgelesen werden soll, ist das Feld `glue_set`, in dem das Maß der Stauchung oder Dehnung einer gesetzten Zeile gespeichert ist. Ist der Wert null, so musste keine Dehnung vorgenommen werden, ist er hingegen 1, wurde der Leerraum um 100% gedehnt oder gestaucht; das Vorzeichen gibt die Verkürzung oder Verlängerung der Zeile an.

Hat man diesen Wert für jede Zeile ausgelesen, kann man ihn in eine Farbdarstellung kodieren (0 entspricht mittelgrau, -1 sehr dunklem grau, +1 sehr hellem) und wie oben einen Farbknoten einfügen. Der Balken selbst ist ein Node vom Typ `rule`, der einfach in die jeweilige horizontale Liste eingefügt wird.

Will man auch den Effekt von Font-Expansion darstellen, also der mikrotypographischen Erweiterung, dass Zeichen leicht gedehnt oder gestaucht werden können, um den Satz zu optimieren, muss man etwas trickreicher arbeiten. Denn es gibt kein Feld, in dem der Wert gespeichert wird – allerdings hat jeder Node das Feld `.width`, das seine Breite angibt. So kann man den jeweils ersten Buchstaben einer Zeile einlesen, dessen Breite abfragen und sie mit der Breite vergleichen, die der Buchstabe in der aktuellen Schrift hat, also nicht gedehnt.

Das Verhältnis der beiden Breiten ist das Maß für die Expansion und kann ebenfalls in einem Balken dargestellt werden. Dieser Absatz wurde mit beiden Darstellungen gesetzt; links ist der Balken für Font Expansion und rechts der für die Leerraumdehnung. Sind die Balken rechts von sehr ähnlicher Farbe, ist der Absatz optimal gesetzt.

Das Paket `chickenize`

Alle hier vorgestellten Funktionen und weitere »Spielereien« sind im Paket `chickenize` zusammengefasst [7] und mit einer einfachen Nutzerschnittstelle versehen. Der Name wurde durch einen Vortrag von Doug Zonkher [8] inspiriert und stellt eine Funktion gleichen Namens zur Verfügung. Diese ersetzt jedes Wort eines beliebigen Dokumentes mit dem Wort »chicken«. Es kann auch nur ein gewisser Anteil zufallsgesteuert ersetzt werden. So kann man etwa den Korrektor eines Textes prüfen – findet er in jedem Korrekturvorgang alle »chicken«-Einträge, hat er sauber gearbeitet.

Das Einfügen ganzer Wörter, inklusive korrekter Trennstellen, Kerning und u. U. auch Ligaturen ist deutlich komplexer als die oben beschriebenen Funktionen und ist noch nicht voll funktionsfähig implementiert; hier sei wieder auf Patrick

Gundlachs System verwiesen. [1] Mit dessen Methoden ist es auch möglich, bestimmte Wörter in der Eingabe zu suchen und zu ersetzen, ohne die Eingabedatei selbst zu ändern – vielleicht ein Ansatz für Serienbrieferstellung.

Insgesamt verfügt das Paket `chickenize` inzwischen über mehrere lehrreiche Funktionen, die alle nur auf Lua-Code basieren und daher sowohl für LaTeX als auch plainTeX geeignet sind; ein Test unter ConTeXt steht noch aus. Unter anderem kann der Nutzer mit einfachen Befehlen folgende Effekte dokumentenweit oder nur in einem beschränkten Bereich erzeugen (Lehrreich für die Verwendung von Attributen):

- `chickenize` – ersetzt jedes Wort durch »chicken«
- `colorstretch` – zeigt Grauwert und Font Expansion als graue Balken
- `leetspeak` – ersetzt alle Zeichen in 1337-Schreibweise
- `randomuclc` – vertauscht zufällig Groß- und Kleinbuchstaben
- `randomchars` – vollkommen zufällige Ausgabe
- `randomcolor` – setzt alle Buchstaben bunt
- `rainbowcolor` – regenbogenartiger Farbverlauf über alle Zeichen
- `uppercasecolor` – farbige Großbuchstaben

Alle Funktionen verfügen über mehrere Parameter, die der Nutzer mit einfacher Syntax beeinflussen kann. Einige weitere Ideen warten noch auf ihre (vollständige) Umsetzung, u. a. ein zufällig gezeichnetes Hühnchen, das nur mittels Lua-Code gemalt wird, ebenfalls inspiriert von einem Artikel von Paul Isambert. [2]

Außerdem soll das Paket eine Sammelstelle für kreative Ideen mit ähnlichen Ansätzen darstellen – Vorschläge zur Verbesserung oder Erweiterung sind sehr willkommen!

Danksagung

Das `chickenize`-Paket und in dessen Folge der Vortrag sowie dieser Artikel wären ohne die Hilfe und Erklärungen von Paul Isambert und Patrick Gundlach nicht möglich gewesen. Ich danke auch Philipp Gesang für hilfreiche Hinweise und hoffe, seine Vorschläge bald noch umsetzen zu können.

Literatur

[1] Patrick Gundlach: *TEX without TEX*; Artikel im LuaTEX-wiki unter `http://wiki.luatex.org/index.php/TeX_without_TeX`.

[2] Paul Isambert: *Drawing tables: Graphic fun with LuaTEX*; In: TUGboat 32:2, 2011, S. 146.

[3] Paul Isambert: *Three things you can do with LuaTeX that would be extremely painful otherwise*; In: TUGboat 31:3, 2010, S. 184.

[4] Markus Kohm und Alexander Willand: *Alles in einem – Texte und Tabellen mit LuaLaTeX*; In: Die TeXnische Komödie, 2/2011, S. 36.

[5] Arno Trautmann: *chickenize – Spaß mit Node-Manipulationen in LuaTeX*; Vortrag auf der DANTE-Herbsttagung 2011. Folien zum Vortrag unter `http://www.dante.de/events/mv45/Programm.html`.

[6] Arno Trautmann: *An overview of TeX, its children and their friends*; mit TeXLive unter `texdoc tex-overview`, und auf `CTAN:info/tex-overview`.

[7] Arno Trautmann: *The chickenize package*; aktueller Paket-Code unter `https://github.com/alt/chickenize`.

[8] Doug Zongker: *Chicken chicken chicken*; Vortrag bei der AAAS humor session, 16. Februar, 2007. Handout zum Vortrag unter `http://isotropic.org/papers/chicken.pdf`.

Ein Programm zur Konvertierung von Artikeln der Wikipedia nach LaTeX

Dirk Hünniger

Es ist aus vielerlei Gründen wünschenswert, eine LaTeX-Quelldatei aus einem Wikipedia-Artikel generieren zu können. Die händische Konvertierung ist sehr aufwändig und fehleranfällig. Es wurde daher ein Compiler für diese Aufgabe entwickelt und für die Betriebssysteme Windows und Linux zum freien Download bereitgestellt. Weiterhin wurde der Quellcode unter einer freien Lizenz veröffentlicht. Das Programm arbeitet auch bei Schwesterprojekten wie beispielsweise Wikibooks.

Einleitung

Einen LaTeX-Export für Wikipedia-Artikel vorzusehen, scheint in vielerlei Hinsicht sinnvoll. Zwar verfügt die Wikipedia über eine Funktion zum PDF-Export, jedoch sind die Möglichkeiten, auf individuelle Wünsche einzugehen, sehr eingeschränkt und daher für Verlage meist nicht hinreichend. Ferner wird die Einbindung von Formeln als Rastergraphiken häufig kritisiert.

Von der Geschichte des Problems bis zum hier vorgestellten Verfahren

In der Vergangenheit gab es bereits mehrere Versuche, das Export-Problem anzugehen. Jedoch gelang es bislang nicht, eine erweiterte Backus-Naur-Form (EBNF) der Wiki-Sprache aufzustellen, die als Grundlage für eine Vielzahl von Compiler-Verfahren notwendig wäre. Es konnte jedoch bislang auch noch nicht gezeigt werden, dass keine solche EBNF existiert, was auch dem Fehlen einer hinreichend formalen Spezifikation der Wiki-Sprache geschuldet sein mag.

Ein vielversprechender Ansatz gelang Hans-Georg Kluge [2], indem er den ursprünglichen Parser der Wiki-Software modifizierte und so die Notwendigkeit einer EBNF umging. Um seine Software sinnvoll betreiben zu können, muss sie jedoch auf den Servern der Betreiber der Wikipedia installiert werden, was bislang nicht geschehen und wohl auch nicht angedacht ist.

Diese Bedingung ist für das hier vorgestellte Programm nicht notwendig, es läuft vollständig auf dem Rechner des Anwenders und fragt lediglich alle benötigten Daten von den Servern[1] der Wikipedia ab. Durch die Parsec-Bibliothek der Programmiersprache Haskell gelang es, die Notwendigkeit einer EBNF zu vermeiden und dennoch den Wiki-Quelltext mit einem unabhängigen Parser analysieren zu können.

Funktionsweise des Systems für den Benutzer

Das Programm erfordert als Eingabe die URL eines Wikipedia-Artikels, führt die notwendigen Arbeiten selbstständig durch und öffnet schließlich eine PDF-Datei welche eine mit LaTeX gesetzte Version des Artikels enthält. Selbstverständlich wird auch der LaTeX-Quelltext mit ggf. eingebundenen Bildern in einer handlichen Form abgelegt.

Derzeit steht die Software für die Betriebssysteme Windows und Ubuntu[2] bereit. Das Projekt ist quelloffen und frei verfügbar.

Durch Manipulation der Header können die für LaTeX üblichen weitgehenden Benutzeranpassungen leicht vorgenommen werden. Als Entsprechung zum `\newcommand`-Befehl von LaTeX gibt es im Wiki-System benutzerdefinierte Vorlagen, sogenannte Templates. Diese werden auf `\newcommand`-Befehle abgebildet, wobei die Details der Abbildung durch den Benutzer konfiguriert werden können.

[1] Insbesondere findet auch der für seine exorbitante Stabilität bekannte Toolserver Verwendung.

[2] Die Verfügbarkeit für Ubuntu impliziert in gewisser Hinsicht die Verfügbarkeit für alle weiteren Unix-Betriebssysteme, insbesondere andere Linux-Distributionen und MacOS, auch wenn die Installation in diesen Fällen ein fundiertes technisches Fachwissen voraussetzt.

Technische Details der Implementierung

Das Programm wurde in zwei Programmiersprachen geschrieben: Python 3.x ist eine objektorientierte Skriptsprache ohne Typenprüfung zur Compile-Zeit. Haskell ist eine rein funktionale Programmiersprache, mit einer aufwändigen Typenprüfung zur Compile-Zeit. Python übernimmt die Aufgaben der Ein- und Ausgabe und Haskell die der eigentlichen Kompilierung und damit den bei weitem aufwändigeren Teil der Arbeit.

Es gelang, die Wiki-Sprache sehr weitgehend zu implementieren. Jedoch wurden Templates nur stark vereinfacht berücksichtigt. Zum Einen weil sie selber eine Programmiersprache bilden, deren vollständige Implementierung sehr zeitaufwändig[3] wäre, zum Anderen weil meist eine individuelle Implementierung des jeweiligen Templates in LaTeX gewünscht wird, welche durch die vorhandene Abbildung von Templates auf `\newcommand`-Befehle auch leicht möglich ist.

Durch geschickte Definition von LaTeX-Makros gelang es, die Notwendigkeit von benutzerseitigen Änderungen im generierten LaTeX-Code auf höchst seltene Fälle zu beschränken. Typischerweise können die benutzerseitigen Änderungen in den Headern erfolgen, welche vom Programm unangetastet bleiben. Die Festlegung der Breite von Bildern und Tabellenspalten kann durch geschickte Ausnutzung der Wiki-Sprache in bequemer Weise direkt im Wiki erfolgen. Es wird das im HTML-Standard festgelegte Attribut `width` ausgewertet. Liegt eine prozentuale Angabe vor, so wird diese als auf die Zeilenlänge bezogen verstanden und entsprechend in den Header der LaTeX-Tabelle eingefügt.

Bei einigen Druckereien ist die maximale Dateigröße beschränkt, daher kann die Bildauflösung der einzubindenden Graphiken in geeigneter Weise herunter gerechnet werden. Für diese sowie alle anderen anfallenden Aufgaben der Bildverarbeitung kommt die ImageMagick-Bibliothek zum Einsatz.

Zusammenfassung

Ein für jeden LaTeX-Benutzer anwendbares Verfahren zur Konvertierung von Wikipedia-Artikeln in LaTeX-Dokumente wurde im vorliegenden Artikel vorgestellt. Derzeit sind keine Weiterentwicklungen des Projektes geplant. Hinweise und Verbesserungsvorschläge sind dennoch natürlich jederzeit herzlich willkommen.

[3] Mangels einer formalen Sprachspezifikation wäre eine derartige Implementierung wohl auch unvollkommen.

Literatur

[1] Dirk Hünniger: *wb2pdf*; Website; 2011; Online verfügbar über `http://de.wikibooks.org/wiki/Benutzer:Dirk_Huenniger/wb2pdf`; besucht am 26. Oktorber 2011.

[2] Hans-Georg Kluge: *Extension:Wiki2LaTeX*; Website; 2011; Online verfügbar über `http://www.mediawiki.org/wiki/Extension:Wiki2LaTeX`; besucht am 26. Oktorber 2011.

Aktuelle experimentelle deutsche Trennmuster unter Debian-TEXLive

Petra Rübe-Pugliese

Es wird erklärt, wie die neuesten experimentellen deutschen Trennmuster auf einem Debian-TEXLive-System installiert werden können, ohne dass diese Konfiguration bei System-Updates überschrieben wird.

Die Ausgangssituation

Da ja bekanntlich »alles fließt«, haben die TEX-Installationen auf meinen Rechnern im Laufe der Zeit viele verschiedene Stadien durchlaufen: Vom csTEX auf dem guten alten Atari ST zu teTEX unter SUSE- und später Debian-Linux; dann ein »externes«, von der DANTE-CD installiertes TEXLive-System, das dem Debian-System gegenüber durch ein Dummy-Package vertreten wurde; und heute ein Basissystem, bestehend aus regulären TEXLive-Paketen aus der Debian-Distribution, ergänzt durch private Aktualisierungen im lokalen `texmf`-Baum.

Über neue Versionen mich interessierender Pakete informieren mich `CTAN-Ann` sowie ein selbstgestricktes Skript, das die Datei `CTAN:FILES.last07days` auf die betreffenden Paketnamen durchsucht.

Das Einspielen der Änderungen ist mit etwas Handarbeit verbunden, die sich aber in vertretbarem Rahmen hält, da die Anzahl der für mich relevanten Updates überschaubar bleibt. Auch die Anzahl der jeweils zu aktualisierenden Dateien ist meistens übersichtlich und auf ein einzelnes Verzeichnis in `/usr/local/share/texmf/tex/latex` begrenzt.

Font-Updates gestalten sich etwas aufwändiger, da sich die Dateien auf viele verschiedene Verzeichnisse verteilen (nein, ich benutze noch kein XETEX!), kommen im Schnitt aber nur alle paar Monate einmal vor.

Die Zweckmäßigkeit dieses Vorgehens, etwa im Vergleich zu dem in dem ausführlichen Aufsatz von Enrico Gregorio über die Installation von TEXLive unter Ubuntu in der letzten DTK [1] vorgestellten Verfahren, will ich an dieser Stelle nicht diskutieren, da das ein Thema für sich ist.

Das Problem

Seit wann ich das `hyphsubst`-Paket [3] aus dem Oberdiek-Bündel verwende, kann ich nicht genau rekonstruieren; jedenfalls beginnen meine LATEX-Dateien seit mindestens zwei Jahren mit:

```
\RequirePackage[ngerman=ngerman-x-latest]{hyphsubst}
\documentclass[...]{scrartcl}
```

Deshalb merkte ich sofort auf, als CTAN-Ann am 2. Juli 2011 die Meldung vom Update des `dehyph-exptl`-Paketes [4] verbreitete. Mir war bewusst, dass es sich dabei um ein Update handelte, das direkt den Kern des LATEX-Systems, nämlich die Format-Dateien, betraf, und dass daher mit Vorsicht vorzugehen war.

In der Tat hatte die zugehörige `INSTALL`-Datei [5] die folgende Warnung für mich parat:

> *Achtung: Die manuelle Installation in TEXLive wird nicht empfohlen, da die Datei* `language.dat` *in der Regel automatisch verwaltet wird, hier jedoch von Hand Änderungen vorgenommen werden. Es kann daher bei späteren (unabhängigen) Updates zu Problemen kommen!*
>
> *In TEXLive-2008 ist dieses Paket bereits vorinstalliert und ohne weitere Vorkehrungen nutzbar.*

Dass experimentelle deutsche Trennmuster in meinen Debian-Paketen (immerhin jetzt schon von 2009 `;-)`) enthalten sind, war mir bewusst – darauf basierte ja meine Nutzung von `hyphsubst` – aber ich wollte doch die *neuesten* Trennmuster haben!

Ein grober Überschlag des Verhältnisses von Aufwand (eventuell erheblich?) zu Effekt (wahrscheinlich kein spürbarer!) führte jedoch dazu, dass ich die Aktualisierung der deutschen Trennmuster an dieser Stelle zunächst einmal abbrach.

Die Lösung

Die Lektüre des Trennmuster-Artikels von Rolf Niepraschk in der letzten DTK [2] führte dazu, dass ich mich erneut mit dem Thema beschäftigte.

Auch hier [2, S. 50] findet sich der Hinweis, dass die neuen Trennmuster (a) in der TEX-Installation enthalten und (b) in das benutzte Format eingebunden sein müssen.

Gleichzeitig erinnerte ich mich, dass es doch eigentlich zu den Grundsätzen von Debian gehört, dass sehr vieles durch den Benutzer konfigurierbar ist und dass diese Benutzereinstellungen bei Updates respektiert und eben *nicht* überschrieben werden sollen. Eine kurze Inspektion der Systemdateien offenbarte dann auch die sehr einfache Lösung des Problems. Tatsächlich lässt sich das angestrebte Ziel mit den folgenden wenigen Schritten erreichen. Dabei müssen alle Schritte außer dem ersten mit Systemverwalterrechten ausgeführt werden, egal ob man sich diese durch Einloggen als `root`, mit `su` oder mit `sudo` verschafft.

1. Die Archivdatei `dehyph-exptl.zip` von CTAN (`CTAN:language/hyphenation/dehyph-exptl.zip`) herunterladen und in einem temporären Verzeichnis entpacken.
2. Erzeugung des Verzeichnisses `/usr/local/share/texmf/tex/generic/dehyph-exptl/`.
3. Verschieben der Dateien
   `dehypht-x-2011-07-01.pat`,
   `dehypht-x-2011-07-01.tex`,
   `dehyphn-x-2011-07-01.pat`,
   `dehyphn-x-2011-07-01.tex`,
   `dehyphts-x-2011-07-01.pat` und
   `dehyphts-x-2011-07-01.tex`,
   die durch Auspacken der `zip`-Datei entstanden sind, nach `/usr/local/share/texmf/tex/generic/dehyph-exptl/`.
4. Aktualisierung der »filename database« mit `mktexlsr` bzw. `texhash` (was beides dasselbe Programm bezeichnet).
5. Editieren der Konfigurationsdatei `/etc/texmf/hyphen.d/10texlive-lang-german.cnf`. Diese Datei sollte vorhanden sein, sofern – was bei einem deutschsprachigen System zu erwarten ist! – das Debian-Paket `texlive-lang-german` installiert ist. Bei etwaigen Unklarheiten über diesen Punkt gebe man auf der Kommandozeile den Befehl `apt-cache policy texlive-lang-german` ein!
   Die Datei sah bei mir im Originalzustand folgendermaßen aus:

```
% 10texlive-lang-german.cnf
% You can change/add entries to this file and changes will be preserved
% over upgrades, even if you have removed the main package prior
% (not if you purged it). You should leave the following pseudo comment
% present in the file!
% -_- DebPkgProvidedMaps -_-
%
```

```
name=ngerman file=loadhyph-de-1996.tex lefthyphenmin=2 righthyphenmin=2
name=german file=loadhyph-de-1901.tex lefthyphenmin=2 righthyphenmin=2
name=german-x-2009-06-19 file=dehypht-x-2009-06-19.tex lefthyphenmin=2
 righthyphenmin=2 synonyms=german-x-latest
name=ngerman-x-2009-06-19 file=dehyphn-x-2009-06-19.tex lefthyphenmin=2
 righthyphenmin=2 synonyms=ngerman-x-latest
```

Man sieht: Das Einzige, was angepasst werden muss, sind die Daten in den beiden letzten Zeilen! Nach den Änderungen sehen diese bei mir nun folgendermaßen aus:

```
name=german-x-2011-07-01  file=dehypht-x-2009-06-19.tex lefthyphenmin=2
 righthyphenmin=2 synonyms=german-x-latest
name=ngerman-x-2011-07-01 file=dehyphn-x-2009-06-19.tex lefthyphenmin=2
 righthyphenmin=2 synonyms=ngerman-x-latest
```

Wer will, kann auch eine entsprechende Zeile für die schweizer Trennmuster (`dehyphts-x-2011-07-01.*`) einfügen.

6. Aus der so editierten Konfigurationsdatei und anderen, entsprechenden Konfigurationsdateien für weitere installierte Sprachen wird nun die `language.dat` erzeugt, vor deren Überschreiben die `dehyph-exptl-INSTALL`-Anleitung gewarnt hatte. Wer diesen Vorgang genau verfolgen will, wirft vor Ausführung des diesbezüglichen Befehls einen Blick auf Datum und Inhalt der Datei `/var/lib/texmf/tex/generic/config/language.dat`.
   Nach Ausführung des Befehls `update-language` müsste sich beides geändert haben.
7. Abschließend müssen mit dieser geänderten `language.dat` die Formate neu erzeugt werden. Das bewerkstelligt der Befehl `fmtutil-sys --all`.

Das war's schon! Am Anfang jedes `latex`-Aufrufs müssten jetzt die in das Format eingebundenen aktualisierten Trennmuster angezeigt werden, z. B.:

```
~/tmp/demo > pdflatex demo.tex
This is pdfTeX, Version 3.1415926-1.40.10 (TeX Live 2009/Debian)
entering extended mode
(./demo.tex
LaTeX2e <2009/09/24>
Babel <v3.8l> and hyphenation patterns for english, usenglishmax, dumylang, noh
yphenation, russian, french, ngerman, german, german-x-2011-07-01, ngerman-x-20
11-07-01, spanish, ukenglish, loaded.
(usw.)
```

Für das jeweilige Dokument verfügbar gemacht werden sie, wie in [2], [3] und [4, S. 6] beschrieben, durch das `hyphsubst`-Paket.

Verhalten bei Updates

Da die einzige Datei, an der wir Änderungen vorgenommen haben, eine Debian-Konfigurationsdatei ist, bleibt diese bei System-Updates unberührt, so lange der Maintainer des `texlive-lang-german`-Paketes an dieser Stelle keine Änderungen ausgeführt hat. Wurde die Datei vom Maintainer geändert, wird der Benutzer gefragt, ob er die von ihm editierte Version behalten, die neue Version einspielen oder zunächst einmal beide Versionen vergleichen will.

Egal was man wählt: Es wird keine Version gelöscht!

Um auf einem einigermaßen aktuellen Stand zu bleiben, wähle ich in einem solchen Fall stets »Version des Package Maintainers einspielen«. Die alte Version bleibt dann, versehen mit der Endung »`.dpkg-old`«, neben der neuen Datei erhalten. Nach Beendigung des Updates vergleiche ich die beiden Versionen (`diff` leistet hier gute Dienste!) und übertrage meine Änderungen – sofern ich sie weiterhin für sinnvoll halte – von der alten in die neue Datei.

In dem vorliegenden Fall müsste dann hinterher noch einmal `update-language`, gefolgt von `fmtutil-sys --all`, ausgeführt werden.

Sollte eines Tages der Fall eintreten, dass die Version der Debian-Trennmuster tatsächlich der neuesten Version entspricht, kann man die neu eingespielte System-Konfigurationsdatei unverändert lassen und das Verzeichnis `/usr/local/share/texmf/tex/generic/dehyph-exptl/` samt Inhalt wieder löschen.

Varianten für andere Systeme?

Unter Ubuntu und anderen Debian-Derivaten müsste es genauso funktionieren. Feedback dazu wäre willkommen!

Literatur

[1] Enrico Gregorio: *Installation der TEXLive 2011 auf Ubuntu*; *Die TEXnische Komödie*; 3, S. 33–45; 2011.

[2] Rolf Niepraschk: *Die experimentellen Trennmuster für LATEX*; *Die TEXnische Komödie*; 3, S. 49–52; 2011.

[3] Heiko Oberdiek: *The hyphsubst package*; Juni 2008; `CTAN:macros/latex/contrib/oberdiek/hyphsubst.pdf`.

[4] Die deutschsprachige Trennmustermannschaft: *dehyph-exptl – Experimentelle Trennmuster für die deutsche Sprache*; Juli 2011; `CTAN:language/hyphenation/dehyph-exptl/dehyph-exptl.pdf`.

[5] Die deutschsprachige Trennmustermannschaft: *Installationshinweise für die experimentellen deutschen Trennmuster*; Juli 2011; `CTAN:language/hyphenation/dehyph-exptl/INSTALL`.

Zu den Nachteilen von BibTeX

Philipp Lehman

Bei den Nachteilen von BibTeX geht es im Allgemeinen um zwei Punkte:

1. BibTeX und auch BibTeX8 beruhen auf der Annahme, dass ein Zeichen von einem Byte repräsentiert wird. Das trifft auf UTF-8 aber nicht zu, wo man für bestimmte Zeichen mehr als ein Byte hat.
2. Es geht aber nicht nur um UTF-8, sondern vor allem um Unicode als umfassenden Standard zur Kodierung von Zeichen sowie um verschiedene andere Dinge, die damit zusammenhängen, wie beispielsweise die Sortierung. Siehe dazu auch `http://unicode.org/reports/tr10/#Common_Misperceptions`.

BibTeX beruht auf dem ASCII-Standard und unterstellt damit in gewisser Hinsicht, dass das englische Alphabet ein universeller Standard sei. Aus diesem Standard wird gleich noch die Sortierwertigkeit der Zeichen abgeleitet, BibTeX gewichtet Zeichen nämlich nach ihrer Position im ASCII. Um zu verstehen, was das bedeutet, empfiehlt sich ein Bilck auf eine ASCII-Tabelle, beispielsweise auf `http://aspell.net/charsets/iso646-us.gif`. Sieht man sich diese Tabelle genauer an, wobei die Reihenfolge von links nach rechts und von oben nach unten geht, stellt man fest, dass der Buchstabe »Z« vor »a« kommt. Und tatsächlich sortiert BibTeX Zeichen in der Reihenfolge 0–9, A–Z, a–z. Um diesen Designfehler zu kaschieren, konvertieren alle `bst`-Dateien die Zeichen vor dem Sortieren in Kleinbuchstaben, sonst würde ein »a« hinter einem »Z« landen.

Schwieriger wird es bei Zeichen mit Umlauten, die im ASCII formal nicht vorhanden sind. Man schreibt dann statt »ä« die TeX-Kodierung »`{\"a}`«. BibTeX sieht eine öffnende Klammer, gefolgt von dem Backslash und ignoriert somit `\"`. Aus Ä, Á, À, Â usw. wird beim Sortieren entsprechend A, aus Ö, Ó, Ò, Ô und Ø wird O, aus Æ wird AE usw. Das entspricht zufällig den Gepflogenheiten im englischen Sprachraum, funktioniert bei anderen Sprachen aber nicht. Im Dänischen beispielsweise ist Å der letzte Buchstabe des Alphabets, Z ist der Viertletzte. Die Sortierreihenfolge lautet: A–Z, Æ, Ø, Å. Im Schwedischen ist Ö der letzte

Buchstabe des Alphabets und Ä der vorletzte. Das kann man vom Deutschen nun wieder nicht behaupten. Dieser gesamte Komplex der Sprachabhängigkeit der Sortierung von Text wird von BibTeX komplett ignoriert.

Um ein Ø richtig zu sortieren, benötigt man eben nicht nur die Möglichkeit, das Zeichen kodierungstechnisch zu verarbeiten, sondern auch eine Regel, die angibt, welche Stellung das Zeichen im Alphabet einer bestimmten Sprache hat. BibTeX verfügt über beides nicht. BibTeX8 deckt zumindest den kodierungstechnischen Aspekt soweit ab, wie die Regel »1 Zeichen = 1 Byte« gilt. In den csf-Dateien kann man die Reihenfolge beliebiger 8-Bit-Zeichen frei festlegen. Das dient in erster Linie dazu, verschiedene 8-Bit-Kodierungen zu unterstützen. Man kann es aber auch nutzen, um eine sprachenspezifische Sortierung der Zeichen zu erhalten, indem man eine csf-Datei für jede Kombination aus Sprache und Kodierung erstellt. Auch das hat Grenzen, weil es beim Sortieren auch zu Ersetzungen kommt, d. h., es kann erforderlich sein, ein Zeichen in mehrere aufzulösen. Im Französischen ist zum Beispiel Œ eine Ligatur, die beim Sortieren in OE aufgetrennt wird, im Deutschen wird ß als »ss« sortiert. Im Dänischen ist Æ hingegen ein eigenständiger Buchstabe, keine Ligatur. Mit BibTeX8 kann man zwar ein Æ im Dänischen richtig sortieren, nicht aber ein Œ im Französischen oder ein ß im Deutschen. BibTeX wiederum kann Œ richtig verarbeiten, sofern es als `{\OE}` notiert ist (was mit BibTeX8 dank Rückwärtskompatibilität natürlich auch geht), scheitert aber an Æ = `{\AE}`, weil »AE« natürlich nicht hinter »Z« landet.

Das ist immer noch nicht alles. Es gibt auch innerhalb einer Sprache beziehungsweise eines Landes teilweise verschiedene Sortierstandards, im Deutschen z. B. DIN 5007-1 und DIN 5007-2 (»Telefonbuchsortierung«): `http://de.wikipedia.org/wiki/Alphabetische_Sortierung`. Bei DIN 5007-2 werden Umlaute aufgelöst; aus »Ö« wird beim Sortieren »Oe« , aus »Ä« wird »Ae« usw. Das ist weder mit BibTeX noch mit BibTeX8 möglich, weil beide auf der Annahme beruhen, dass sich die Wortlänge bei der Sortierung nicht verändert, auch nicht bei der Konvertierung zwischen Groß- und Kleinschreibung (im Deutschen z. B. »ß« zu »SS«). Das heißt konkret: BibTeX ist nicht imstande, ein deutsches Telefonbuch korrekt zu sortieren. Mit BibLaTeX+Biber ist das durch die Wahl der so genannten »Lokalisierung« möglich:

```
\usepackage[backend=biber,sortlocale=de_DE_phonebook]{biblatex}|
```

Es geht wohlgemerkt um die Abhängigkeit der Sortierung von Sprache, Region und weiteren Regeln (hier: deutsch/Deutschland/DIN5007-2), nicht um die Kodierung der Zeichen in der `bib`-Datei (beispielsweise UTF-8), die natürlich auch noch zu berücksichtigen ist.

Ganz ähnliche Probleme gibt es übrigens auch bei Index-Prozessoren.

LaTeX3-Roadmap[1]

Joseph Wright

Ein Frage, die von Zeit zu Zeit aufkommt und mir als Mitglied des Entwicklungsteams öfters gestellt wird, ist die nach dem »Fahrplan« der LaTeX3-Entwicklung. Obwohl es keinen offiziellen Plan gibt, habe ich sicher einige Ideen, die ich gerne konkret angehen würde. Das ist alles ziemlich dynamisch, aber ich werde versuchen, einige wichtige Dinge hervorzuheben, welche Aufmerksamkeit verdienen.

Kurzfristig

Kurzfristig ist über Dinge zu sprechen, von denen LaTeX3 nicht entscheidend abhängt und für die eine Menge Arbeit bereits getan worden ist. Es gibt sehr viele sich entwickelnde Gebiete: Kleine Verbesserungen wirken sich auf eine Vielzahl von Themen aus. Diese Dinge fügen sich alle in das Konzept von »LaTeX3 in LaTeX 2_ε« ein, wo dieses Material ohne Auswirkung auf existierende Dokumente geladen werden kann.

Fließkommaausdrücke

Die aktuelle Fließkomma-Implementierung in LaTeX3 basiert auf der Ausführung von Zuweisungen und ist ziemlich schlicht in der Art und Weise, Werte zu speichern. Bruno Le Floch arbeitet gerade an einem verbesserten Ansatz, der es erlauben wird, Fließkommaausdrücke detailliert zu analysieren. Das wird sehr nützlich für Dinge wie dem Drehen von Boxen sein und zukünftig auch ti*k*Z - ähnliche zeichnerische Fähigkeiten ermöglichen.

Erweiterte `key-value`-Konzepte

Das aktuelle `l3keys`-Modul basiert lose auf den `pgfkeys`. Ein Bereich, der nicht implementiert ist, ist die Idee des »key path«. Die Benutzung eines Pfadmodells, um Schlüssel zu setzen, hat einige Vorteile. Deshalb würde ich gerne etwas hinzufügen, das Folgendes ermöglicht:

```
\keys_set:n
  {
    /module .cd ,
    key-a = value ,
    key-b .meta = { key-a = setting , /other-module/key = value }
  }
```

[1] Übersetzt von Roland Geiger. Mit Dank an Frauke Schmidt für hilfreiche Diskussionen und Kommentare bei der Übersetzung.

LaTeX3-Fehlerchen

Es gibt eine Reihe von Punkten, die aufgekommen sind, seit LaTeX3 mehr benutzt wird. Diese zu reparieren ist eine ständige Aufgabe, und wir werden uns wahrscheinlich um einige in den kommenden Wochen kümmern. Der Nachschub an Problemen wird uns nicht ausgehen!

Verbesserung der Dokumentation

Ein Hauptziel für LaTeX3 ist es, den gesamten Code, der von anderen benutzt werden wird, zu dokumentieren. Eine gute Dokumentation zu bekommen ist schwer, und so ist eine andere sich ständig entwickelnde Aufgabe, die Dokumentation zu verbessern. Wir haben bereits ein Stück der Arbeit daran getan, aber sie ist bestimmt noch nicht komplett.

Mittelfristig

Mittelfristig geht es um größere Ideen, aber es gibt einige, zu denen Code geschrieben wurde und deren Ergebnisse innerhalb der nächsten paar Monate erscheinen werden. Hier entfernen wir uns dann mit dem Code von »LaTeX3 in LaTeX 2_ε«, was existierende Dokumente zerstören wird, wenn sie nicht angepasst werden.

Die `Galley`

Die `galley` (Druckfahne) ist der vertikale Bereich, in den der Text und anderer Inhalt platziert ist. Die Verwaltung der `galley` ist die Grundlage in LaTeX 2_ε, und kann zu seltsamen Effekten im vertikalen Vorschub führen. Gibt man beispielsweise

```
\end{itemize}
\vspace{3pt}
\begin{itemize}
```

ein, wird man 12 pt Vorschub erhalten! Wir haben bereits drei `galley`-Implementierungen, und wir arbeiten jetzt an einer vierten. Hier ist zu entscheiden, was der beste Ansatz ist: Die unterschiedlichen Versionen haben alle unterschiedliche Komplexitätsebenen. Wir wollen hoffen, dass wir diese Ideen in den nächsten paar Monaten in einem einzigen funktionierenden Modul lösen können.

Fontauswahl

Das »New Font Selection Scheme« in LaTeX 2_ε ist ein ziemlich guter Ansatz zur Auswahl von Schriften. Als ersten Schritt würde man den Code mehr oder weniger in die LaTeX3-Syntax konvertieren. Es gibt dann ein paar Punkte, über die

man sich Gedanken machen muss, zum Beispiel wie man mit Kapitälchen umgeht. Auch das Laden der Schriften muss angepasst werden. Man benötigt eine Zusammenarbeit von `fontspec` (für XeTeX und LuaTeX) mit dem existierenden pdfTeX-Mechanismus, und es stellt sich die Frage, wie man das erreicht.

Langfristig

Langfristig geht es wieder um große Ideen, aber hier geht es um solche, deren Code noch nicht geschrieben ist. Einige dieser Konzepte sind realisiert, aber es bleibt noch viel zu tun.

Die Ausgaberoutine

Die größte Einzelherausforderung für LaTeX3 ist es, eine gute Ausgaberoutine zu implementieren. Hier haben wir `xor`, einen von Frank Mittelbach geschriebenen Ansatz. Der funktioniert, aber er braucht eine gründliche Überarbeitung, da der Code über viele Jahre entwickelt wurde. So ist er im Augenblick ein ziemlicher Mix aus LaTeX 2_ε- und LaTeX3-Zeilen. Wir müssen auch diskutieren, wie die Ausgaberoutine arbeiten soll, wenn man etwa an den Textfluss in Gestaltungsrastern und bei Registerhaltigkeit denkt.

Jenseits der Ausgaberoutine

Wenn wir all das oben Genannte angegangen haben, dann geht es daran, ein Format zu bekommen, das zum Textsatz fähig ist. Das bedeutet, die Elemente, nämlich die Code-Ebene zum Programmieren, Galley-Dinge, Fonts und Ausgaberoutine zu einem eigenständigen Format zu integrieren. Das Ergebnis wird erstmal nicht beeindrucken, da es noch nicht viel der üblichen Markup-Möglichkeiten bietet. Jedoch wird es das Testen ohne LaTeX 2_ε ermöglichen. Das ist dann der Zeitpunkt, wo wir Dinge wie die Gliederungsbefehle, Umgang mit Gleitobjekten und all die anderen Sachen hinzufügen müssen, mit denen das jetzige LaTeX umgehen kann.

Gepostet in LaTeX3 `http://www.texdev.net/2011/09/05/latex3-roadmap/`

Berichtigung zu »Von \pageref zu \hyperpage«

Herbert Möller

Beim Einsatz des Pakets hyperref mit pdfTEX ergeben die aus \pageref erzeugten Links oft fehlerhafte Sprünge. [1] enthält ein Perl-Filterprogramm, mit dessen Hilfe sich dieses Problem lösen lässt. Dazu wird aus den mit \newlabel beginnenden Zeilen der betreffenden .aux-Dateien ein neues Filterprogramm aufgebaut, das dann in den zugehörigen .tex- oder .ltx-Dateien die Ersetzungen von \pageref mit der symbolischen Adresse durch \hyperpage mit der entsprechenden Seitenzahl durchführt

Bei dem Austausch der Klammern mit der Abschnittskennzeichnung wurden irrtümlich einige Möglichkeiten nicht berücksichtigt. Das folgende Filterprogramm funktioniert nun auch bei Labelpositionen vor dem ersten Buchkapitel oder Artikelabschnitt, bei beliebigen Gliederungstiefen und in Anhängen.

```
#!/usr/bin/perl -w
# Auxnew.pl
@lines = <>;
$pert = "#!/usr/bin/perl -w\n# Pagex.pl\n  undef \$/;\n  \$_ = <>;\n";
do {
  $_ = $lines[$i++];
  if (/newlabel/) {
    s/\\newlabel/s\/pageref\\/o;
    s/\}\{\{.*\}\{(\d+)\}.+$/\\\}\/hyperpage\\\{$1\\\}\/go;/o;
    $pert .= $_;
  }
}
until ($i == $#lines);
$pert .= "print STDOUT;\n";
print $pert;
```

Die Anwendung des Perl-Systems unter den verbreiteten Betriebssystemen wurde in [2] beschrieben. Von Markus Kaupp kam der Hinweis, dass der (kostenlos erhältliche) Texteditor »Komodo Edit 5.1« unter 64-Bit Windows 7 den ausgewählten Text nicht an das Filterprogramm übergibt. Stattdessen kann Letzteres über die normale Windows-Kommandozeile aufgerufen und die zu filternde Datei als Parameter übergeben werden.

Literatur

[1] Herbert Möller: *Von \pageref zu \hyperpage*; Die TEXnische Komödie; 4/2009.

[2] Herbert Möller: *LaTeX-Figuren mit Hilfe von OpenOffice.org 3 Draw*; Die TeXnische Komödie; 4/2009.

Anmerkung von Heiko Oberdiek

Die Methode lässt sich bereits auf TeX-Ebene realisieren:

```
\usepackage{nameref}
\usepackage{hyperref}
\usepackage{refcount}[2010/12/01]
\usepackage{zref}% hier nur fuer \zref@wrapper@babel

\makeatletter
\AtBeginDocument{%
  \let\SavedPageref\pageref
  \renewcommand*{\pageref}{%
    \@ifstar{%
      \SavedPageref*%
    }{%
      \zref@wrapper@babel{\myhyperpageref}%
    }%
  }%
  \newcommand*{\myhyperpageref}[1]{%
    \refused{#1}%
    \IfRefUndefinedExpandable{#1}{%
      \SavedPageref{#1}%
    }{%
      \hyperpage{\getpagerefnumber{#1}}%
    }%
  }%
}
\makeatother
```

Die Methode identifiziert die Seiten via `\thepage`, Seitenanker dürfen nicht abgestellt sein und müssen über `\thepage` adressiert werden (`hyperref`-Voreinstellung). Das bedeutet, `\thepage` muss eine Seite eindeutig identifizieren. Besser wäre eine Identifizierung über die absolute Seitennummer. Aber das gibt das Referenzsystem von LaTeX leider nicht her und man braucht so etwas wie `zref`.

Von fremden Bühnen

Neue Pakete auf CTAN

Jürgen Fenn

Der Beitrag stellt neue Pakete auf CTAN seit der letzten Ausgabe bis zum Redaktionsschluss vor. Die Updates können auf der *ctan-ann*-Mailingliste verfolgt werden, die auch über Twitter und Identi.ca als `@ctanannounce` verfügbar sind.

calligra-type1 von *Khaled Hosny* ist eine Version des Fonts `Calligra` von *Peter Vanroose*.

`CTAN:fonts/calligra-type1`

business-research von *Andreas Löffler* und *Sebastian Schanz* ist eine LaTeX-Klasse für die Zeitschrift *Business Research* des *Verbands der Hochschullehrer für Betriebswirtschaft*.

`CTAN:macros/latex/contrib/business-research`

translation-enumitem-de von *Christine Römer* und *Matthias Ludwig* ist die deutsche Übersetzung der Anleitung zu dem Paket `enumitem`.

`CTAN:info/translations/enumitem/de`

tqft von *Andrew Stacey* dient zum Zeichnen von Diagrammen zur topologischen Quantenfeldtheorie (TQFT) mit Hilfe von pgf/TikZ.

`CTAN:macros/latex/contrib/tqft`

xecolor von *Vafa Khalighi* dient zum Definieren von 140 verschiedenen Farben, die in XeTeX auch zum Setzen von bidirektionalen Texten verwendet werden können.

`CTAN:macros/xetex/latex/xecolor`

rtsched von *Giuseppe Lipari* dient zum Zeichnen von Gantt-Diagrammen, die im Rahmen des Projektmanagements verwendet werden.

`CTAN:macros/latex/contrib/rtsched`

moderntimeline von *Raphael Pinson* ergänzt die Einträge bei Lebensläufen, die mit dem Paket `moderncv` erstellt werden, um eine Timeline.

`CTAN:macros/latex/contrib/moderntimeline`

modiagram von *Clemens Niederberger* dient zum Zeichnen von Molekülorbitaldiagrammen (MO-Diagramme) mit Hilfe von pgf/TikZ und den LaTeX3-Paketen `expl3`, `l3kernel` und `l3packages`.

`CTAN:macros/latex/contrib/modiagram`

storebox von *Martin Scharrer* bietet einen Ersatz für die übliche *save box* für `pdfLaTeX` und `luaLaTeX`. Der Inhalt der Box wird nur einmal in die Ausgabedatei geschrieben, auch wenn er mehrfach verwendet wird.

`CTAN:macros/latex/contrib/storebox`

pgfmolbio von *Wolfgang Skala* dient zum Zeichnen von Grafiken, die man typischerweise in Texten aus dem Bereich

der Molekularbiologie findet.
`CTAN:macros/luatex/latex/pgfmolbio`

biblatex-musuos von *Tobias Weh* ist ein `biblatex`-Stil zur Verwendung mit der Klasse `musuos` für Musikwissenschaftler der Universität Osnabrück.
`CTAN:macros/latex/contrib/biblatex-contrib/biblatex-musuos`

tabfigures von *Michael Ummels* enthält einige Fehlerbereinigungen zum vertikalen Setzen von Ziffern.
`CTAN:macros/latex/contrib/tabfigures`

fontaxes von *Michael Ummels* erweitert das LaTeX New Font Selection Scheme um weitere Font-Achsen.
`CTAN:macros/latex/contrib/fontaxes`

texliveonfly von *Saitulaa Naranong* ist ein Pythonskript, mit dem in TeXLive 2010 und 2011 fehlende Pakete *»on the fly«* heruntergeladen und installiert werden können. Es wurde auf Ubuntu getestet.
`CTAN:support/texliveonfly`

makeglossariesgui von *Nicola Talbot* ist ein graphisches Java-Interface zu dem Perlskript `makeglossaries`, das vor allem für solche Anwender entwickelt wurde, die das Skript nicht von der Kommandozeile aufrufen möchten oder können.
`CTAN:support/makeglossariesgui`

chscite von *Simon Sigurdhsson* ist ein BibTeX-Stil für die *Chalmers University of Technology*.
`CTAN:macros/latex/contrib/chscite`

translation-filecontents-de von *Christine Römer* und *Ronny Berndt* ist die deutsche Übersetzung des Pakets `filecontents`.
`CTAN:info/translations/filecontents/de`

paracol von *Hiroshi Nakashima* ist ein weiteres Paket, mit dem man Text »parallel« nebeneinander in zwei Spalten setzen kann, z. B. Übersetzungen.
`CTAN:macros/latex/contrib/paracol`

delim von *Stefan Majewsky* vereinfacht die Verwendung von Klammern im Mathematiksatz.
`CTAN:macros/latex/contrib/delim`

latex-pro-pragmatiky von *Pavel Satrapa* ist eine LaTeX-Einführung auf Tschechisch.
`CTAN:info/czech/latex-pro-pragmatiky`

ifoddpage von *Martin Scharrer* stellt Makros bereit, um zu testen, ob die aktuelle Seite des Dokuments eine gerade oder eine ungerade Seitennummer hat.
`CTAN:macros/latex/contrib/ifoddpage`

dhua von *Uwe Lück* bietet Befehle für mehrgliedrige deutsche Abkürzungen, für die schmale Leerzeichen (Spatien, Befehl: `\,`) empfohlen werden.
`CTAN:macros/latex/contrib/dhua`

tikzpagenodes von *Martin Scharrer* stellt zusätzliche TikZ-Nodes für den Textbereich, die Kopf- und Fußzeilen sowie für die Randnotizen der aktuelle Seite bereit.
`CTAN:graphics/pgf/contrib/tikzpagenodes`

translation-evc-de von *Christine Römer* und *Katrin Vogel* ist die deutsche Übersetzung der Anleitung zu `ecv`.
`CTAN:info/translations/ecv/de`

moreverb-de von *Christine Römer* und *Matthias Biniok* ist die deutsche Übersetzung der Anleitung zu `moreverb`.
`CTAN:info/translations/moreverb/de`

tikz-cd von *Florencio Neves* dient zum Zeichnen von kommutativen Diagrammen mit Hilfe von pgf/TikZ.
`CTAN:graphics/pgf/contrib/tikz-cd`

impnattypo von *Raphaël Pinson* setzt die typographischen Empfehlungen der französischen *Imprimerie Nationale* um.
`CTAN:macros/latex/contrib/impnattypo`

facture von *Maïeul Rouquette* dient zum Erstellen von einfachen Rechnungen, vorläufig nur in französischer Sprache.
`CTAN:macros/xetex/latex/facture`

coseoul von *Michael Teubner* stellt Befehle bereit, mit denen die Gliederungsebenen eines Textes »relativ« angegeben werden können.
CTAN:macros/latex/contrib/coseoul

xcite von *Enrico Gregorio* ermöglicht es, auf die Schlüssel von Zitaten aus einem anderen Dokument zuzugreifen, analog zu den *labels* von Querverweisen mit dem Paket `xr`.
CTAN:macros/latex/contrib/xcite

lapdf von *Detlef Reimers* erlaubt es, farbige Zeichnungen mit PDF-Primitiven mittels pdfLaTeX zu erzeugen.
CTAN:macros/latex/contrib/lapdf

biblatex-nejm von *Marco Daniel* ist ein `biblatex`-Stil für das *New England Journal of Medicine*.
CTAN:macros/latex/contrib/biblatex-contrib/biblatex-nejm

translation-chemsym-de von *Christine Römer* und *Sandro Heuke* ist die deutsche Übersetzung der Anleitung zu `chemsym`.
CTAN:info/translations/chemsym/de

translation-arsclassica-de von *Christine Römer* und *Iona Gessinger* ist die deutsche Übersetzung der Anleitung zu `arsclassica`.
CTAN:info/translations/arsclassica/de

gitinfo von *Brent Longborough* fügt Git-Metadaten in LaTeX-Dokumente ein.
CTAN:macros/latex/contrib/gitinfo

europecv-de von *Christine Römer* und *Susanne Fischer* ist die deutsche Übersetzung der Anleitung zu `europecv`.
CTAN:info/translations/europecv/de

serbian-def-cyr von *Zoran Filipovic* stellt Gliederungsbefehle in serbischer Sprache für kyrillische Schriftzeichen in T2A-Kodierung und cp1251-Codepage zur Verfügung.
CTAN:macros/latex/contrib/serbian-def-cyr

tagging von *Brent Longborough* erlaubt es, Teile eines Dokuments zu markieren, die einbezogen oder ausgelassen werden sollen, um mehrere Versionen aus derselben Quelle setzen zu können.
CTAN:macros/latex/contrib/tagging

rviewport von *Boris Veytsman* erweitert das Paket `graphicx` um den neuen Schlüssel *rviewport*, um die Größe von Grafiken in Bruchteilen (»relativ«) an das Paket übergeben zu können.
CTAN:macros/latex/contrib/rviewport

persian-modern von *Vafa Khalighi* ist eine Schriftfamilie, die auf den *FarsiTeX Scientific fonts* beruht und zwölf Textfonts umfasst.
CTAN:fonts/persian-modern

braids von *Andrew Stacey* dient zum Zeichnen von Braid-Diagrammen mit Hilfe von pgf/TikZ.
CTAN:graphics/pgf/contrib/braids

bickham von *Michael Sharpe* enthält die virtuellen Fonts und die LaTeX-Unterstützung für die Schriftart *Adobe Bickham Script Pro* als kalligraphischen Mathematikfont (magere und fette Schnitte).
CTAN:fonts/bickham

opensans von *Mohamed El Morabity* enthält die LaTeX-Unterstützung für den gleichnamigen TrueType-Font aus dem Google Font Directory.
CTAN:fonts/opensans

sidenotes von *Andy Thomas* erlaubt es, Text mit Anmerkungen, Abbildungen und Tabellen in den Seitenrand zu setzen.
CTAN:macros/latex/contrib/sidenotes

musuos von *Tobias Weh* ist eine Klasse für Arbeiten am Fachbereich Musik der Universität Osnabrück.
CTAN:macros/latex/contrib/musuos

nlatexdb von *Robin Hoens* ist kompatibel zu den MySQL-LaTeX-Paketen `latexdb` und

`ratexdb` mit ein paar Verbesserungen.
`CTAN:support/nlatexdb`

pxtxalfa von *Michael Sharpe* ist ein virtueller Font, der Buchstaben für den Mathematiksatz bereitstellt und der auf `pxfonts` und `txfonts` beruht.
`CTAN:fonts/pxtxalfa`

babyloniannum von *Raphaël Pinson* dient zum Setzen babylonischer Ziffern mit Hilfe von XeTeX und der Schriftart *Santakku Paleo-Babylonian*.
`CTAN:macros/xetex/latex/babyloniannum`

keyval2e von *Ahmed Musa* ist ein schlanker und robuster Key-Value-Parser.
`CTAN:macros/latex/contrib/keyval2e`

dejavu von *Pavel Farar* enthält den TrueType- und Type1-Font DejaVu, der auf der Schriftart Vera beruht, einschließlich der LaTeX-Unterstützung.
`CTAN:fonts/dejavu`

quoting von *Thomas Titz* bietet eine Variante der Standardumgebungen `quote` und `quotation`, die über ein Key-Value-Interface gesteuert wird.
`CTAN:macros/latex/contrib/quoting`

realboxes von *Martin Scharrer* beruht auf `collectbox` und stellt Varianten der `\box`-Makros bereit.
`CTAN:macros/latex/contrib/realboxes`

pagecolor von *H.-Martin Münch* dient zum Festlegen der Hintergrundfarbe für eine Seite.
`CTAN:macros/latex/contrib/pagecolor`

bhcexam von *Charles Bao* ist eine Klasse für den Mathematikunterricht an chinesischen weiterführenden Schulen.
`CTAN:macros/latex/contrib/bhcexam`

mpcolornames von *Stephan Hennig* definiert mehr als 500 Farbnamen in diversen Farbräumen in METAPOST, einschließlich X11, SVG, DVIPS und `xcolor`.
`CTAN:graphics/metapost/contrib/macros/mpcolornames`

macros2e von *Martin Scharrer* bietet einen (derzeit noch nicht ganz vollständigen) Überblick über die internen Makrodefinitionen von LaTeX 2_ε mit Hyperlinks zu ihren jeweiligen Beschreibungen in `source2e`.
`CTAN:info/macros2e`

biblatex-juradiss von *Tobias Schwan* ist ein `biblatex`-Stil für juristische Arbeiten, der auf `biblatex-dw` aufbaut.
`CTAN:macros/latex/contrib/biblatex-contrib/biblatex-juradiss`

jvlisting von *Jochen Voss* stellt die Umgebung `listing` bereit, in der Quelltext in ein Dokument eingefügt werden kann. Leerzeichen zu Beginn einer Zeile bleiben dabei erhalten.
`CTAN:macros/latex/contrib/jvlisting`

collectbox von *Martin Scharrer* speichert die Argumente von Makros in Boxen, um sie in anderen Makros weiter verwenden zu können.
`CTAN:macros/latex/contrib/collectbox`

ltxkeys von *Ahmed Musa* ist ein Schlüsselparser, der robuster und schneller ist als die Pakete `keyval` und `xkeyval`. Die Klammern in Schlüsselwerten bleiben beim Parsen erhalten.
`CTAN:macros/latex/contrib/ltxkeys`

morewrites von *Bruno Le Floch* erhöht die Beschänkung für Ausgabestreams durch Hinzufügen von Hooks zu `\immediate`, `\openout`, `\write` und `\closeout`.
`CTAN:macros/latex/contrib/morewrites`

moreenum von *Seamus Bradley* erweitert die Umgebung `enumerate` um griechische Buchstaben sowie hexadezimale, binäre und englische Ordinalziffern.
`CTAN:macros/latex/contrib/moreenum`

interpreter von *Paul Isambert* ist ein Paket, das Eingabedateien, die Lua *regular expressions* enthalten, für Lua(La)TeX vorbearbeitet, indem es sie in (La)TeX übersetzt.
`CTAN:macros/luatex/generic/interpreter`

Bücher und Rezensionen

Edition dante – Neuerscheinung

Herbert Voß:

Einführung in LaTeX; 1. Auflage 2011,
DANTE e. V. und Lehmanns Media
943 Seiten; ISBN 978-3-86541-415-1;
29,95 € (Ladenpreis) bzw. 25,– € für Mitglieder
von DANTE e. V., jeweils versandkostenfrei.

Bestellung

Bitte schicken Sie eine E-Mail an `office@dante.de` mit Angabe von *Name, Anschrift, Mitgliedsnummer* und *Anzahl der Exemplare*, und überweisen Sie den Betrag auf das Konto von DANTE e.V. oder bezahlen Sie per PayPal. Die Kontonummer finden Sie am Ende dieses Heftes und Informationen zu PayPal auf `http://www.dante.de/index/Intern/Zahlung.html`.

Bitte beachten Sie für Bestellungen bei DANTE e. V. folgende Informationen zum Widerrufsrecht: Verbraucher können bei Bestellungen per E-Mail, Internet, Brief oder Telefon den Kaufvertrag innerhalb einer Frist von 14 Tagen ab Erhalt der Ware per Brief, Fax oder E-Mail oder durch Rücksendung der Ware widerrufen (siehe Kontaktadresse). Zur Wahrung der Frist genügt die rechtzeitige Absendung des Widerrufs oder der Ware. Bei einem Bestellwert bis 40,– € hat der Besteller die Rücksendekosten zu tragen. Bei Verschlechterung der Ware, die über die übliche Prüfung der Ware hinausgeht, hat der Besteller gegebenenfalls Wertersatz zu leisten.

Leserbriefe

Anmerkungen zu »Viele Ziele – Multi Target Publishing« von Axel Kielhorn

Dominik Wagenführ

Ich kenne die Probleme, die eine Veröffentlichung von LaTeX-Code in mehreren Formaten mit sich bringen. So schreiben/setzen wir »freiesMagazin« in LaTeX. Als Ausgabe haben wir aber nicht nur PDF, sondern auch eine HTML-Version (primär gedacht für Mobilgeräte). Allein die Erzeugung einer HTML-Version war schon ein Akt für sich, war dann aber im Endeffekt mit TTH (`http://hutchinson.belmont.ma.us/tth/`) einigermaßen erfolgreich. Alle anderen TeX-to-HTML-Konverter haben den Dienst versagt, weil sie mit der »Komplexität« des Magazins nicht zurechtgekommen sind.

Die Rufe nach EPUB werden aber in der letzter Zeit bei unseren Lesern immer lauter, sodass wir auch dafür wieder eine Möglichkeit suchen, dieses automatisch zu erzeugen. Pandoc hat dabei leider ähnlich wie andere Konverter versagt. Selbst das von TTH erzeugte HTML kann Pandoc (noch) nicht verarbeiten.

Aus dem Grund untersuchen wir eher Lösungen mit Calibre (`http://calibre-ebook.com/`), um ein EPUB zu erstellen. So versteht Calibre als Eingabe zwar leider kein TeX, aber zumindest Formate wie PDF oder HTML. Aus diesen kann man dann einigermaßen leicht ein EPUB erstellen. Leider funktioniert die Konvertierung auch nur so gut, wie die Ausgangsdatei ist, was in unserem Fall (zumindest für unsere Ansprüche) noch keine perfekten Ergebnisse liefert.

Dennoch ist Calibre sicherlich eine Alternative zu Pandoc, zumal man dann auch den vollen LaTeX-Sprachumfang nutzen kann, um beispielsweise zuerst ein PDF- oder HTML-Dokument zu erstellen.

Spielplan

Termine

2012

7. 3. – 9. 3.	**DANTE 2012** und 46. Mitgliederversammlung von DANTE e.V. Hochschule für Technik, Wirtschaft und Kultur Lipsius-Bau, Karl-Liebknecht-Straße 145, 04277 Leipzig `http://www.dante.de/events/dante2012.html`
17. 3. – 18. 3.	**Chemnitzer Linuxtage 2012** Technische Universität Chemnitz 09107 Chemnitz `http://www.chemnitzer.linux-tage.de`
29. 4. – 3. 5.	**20. BachoTEX-Konferenz 2012** Bachotek, nahe Brodnica, Polen `http://www.gust.org.pl/bachotex/2012`
23. 5. – 26. 5.	**Linuxtag Berlin** Messegelände 14055 Berlin `http://www.linuxtag.org/2012/`
16. 7.–18. 7.	**TUG 2012** Boston, Massachusetts (USA) `http://tug.org/tug2012/`
8. 10. – 12. 10.	**EuroTEX 2012**, 6th International ConTEXt meeting und DANTE-Herbsttagung mit 47. Mitgliederversammlung Breskens (Niederlande) `http://meeting.contextgarden.net/2012/`

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`
Ort derzeit wechselnd
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

Darmstadt
Karlheinz Geyer, `geyerk.fv.tu@nds.tu-darmstadt.de`, `http://www.da-tex.org`
Wechselnder Ort
Erster Freitag im Monat, ab 19.30 Uhr

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 »moschmosch«, Wilhelm-Leuschner-Straße 78, 60329 Frankfurt
Vierter Donnerstag im Monat, 19.30 Uhr

Hamburg
Lothar Fröhling, `lothar@thefroehlings.de`
Restaurant Sandstuv, Neue Straße 17, 21073 Hamburg-Harburg
Letzter Dienstag im Monat, 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.: (06203) 40 22 03, `moss@moss.in-berlin.de`
Anmeldeseite zur Mailingliste: `http://mailman.moss.in-berlin.de/mailman/listinfo/stammtisch-hd-moss.in-berlin.de`
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
Helmut Siegert
Institut für Kristallographie, Zülpicher Straße 49b, 50674 Köln
Letzter Dienstag im Monat, 19.30 Uhr

Konstanz
Kurt Lidwin, kurt.lidwin@web.de,
Restaurant Radieschen, Hohenhausgasse 1, 78462 Konstanz
Zweiter Dienstag im Monat, 19.00 Uhr

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

Stuttgart
Bernd Raichle, bernd.raichle@gmx.de
Bar e Ristorante »Valle«, Geschwister-Scholl-Straße 3, 70197 Stuttgart
Zweiter Dienstag im Monat, 19.30 Uhr

Trier
Martin Sievers,
stammtisch-trier@texberatung.de
Fetzenkneipe (Haus Fetzenreich), Sichelstraße 36 (beim Sieh-Um-Dich), 54290 Trier
Dritter Montag des Monats, 20.15 Uhr

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
BLZ 670 900 00
Kontonummer 2 310 007
IBAN DE67 6709 0000 0002 3100 07
SWIFT-BIC GENODE61MA2

Präsidium

Präsident:	Volker RW Schaa	president@dante.de
Vizepräsident:	Adelheid Grob	vice-president@dante.de
Schatzmeister:	Klaus Höppner	treasurer@dante.de
Schriftführer:	Manfred Lotz	secretary@dante.de
Beisitzer:	Bernd Raichle	
	Martin Sievers	
	Herbert Voß	
	Uwe Ziegenhagen	

Server

CTAN: http://mirror.ctan.org/
DANTE: http://www.dante.de/

FAQ

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

Autoren/Organisatoren

Jürgen Fenn [73]
Friedensallee 174/20
63263 Neu-Isenburg
juergen.fenn@gmx.de

Roland Geiger [67]
Heiterblickallee 4
04329 Leipzig

Patrick Gundlach [24]
Eisenacher Straße 101
10781 Berlin
patrick@gundla.ch

Dirk Hünniger [58]
Krefelder Straße 36
47506 Neukirchen-Vluyn

Gerrit Imsieke [9]
le-tex
Weißenfelser Straße 84
04229 Leipzig

Stefan Kottwitz [16]
Hamburg
stefan@texblog.net

Philipp Lehman [66]
plehman@gmx.net

Manfred Lotz [6]
Schriftführer von DANTE e.V.
manfred@dante.de

Herbert Möller [71]
Elsternweg 10
48329 Havixbeck
mollerh@math.uni-muenster.de

Heiko Oberdiek [24, 72]
Kroppenstück 9
77880 Sasbach
heiko.oberdiek@googlemail.com

Manuel Pégourié-Gonnard [24]
Institut de mathématiques de Jussieu
mpg@elzevir.fr

Petra Rübe-Pugliese [24, 61]
Meiningenallee 3
14052 Berlin
prp@prp.in-berlin.de

Volker RW Schaa [9]
s. Seite 82

Arno Trautmann [50]
Boxbergring 10
69126 HD-Boxberg
arno.trautmann@gmx.de

Herbert Voß [3, 77]
Wasgenstraße 21
14129 Berlin
herbert@dante.de

Dominik Wagenführ [78]
dwagenfuehr@freiesmagazin.de

Joseph Wright [67]
joseph.wright@morningstar2.co.uk

Uwe Ziegenhagen [11]
Köln
uwe@ziegenhagen.info

Die TEXnische Komödie

23. Jahrgang Heft 4/2011 November 2011