Die TEXnische Komödie

DANTE
Deutschsprachige
Anwendervereinigung TEX e.V.

David Kastrup, Markus Kohm, Torsten Krüger, Michael Niedermair, Rolf Niepraschk: *εχTEX – ein Überblick*, Die TEXnische Komödie 4/2003, S. 32-38.

Die TEXnische Komödie ist die Mitgliedszeitschrift von DANTE, Deutschsprachige Anwendervereinigung TEX e.V. Einzelne Hefte können von Mitgliedern bei der Geschäftsstelle von DANTE, Deutschsprachige Anwendervereinigung TEX e.V. erworben werden. Mitglieder erhalten Die TEXnische Komödie im Rahmen ihrer Mitgliedschaft.

$\varepsilon_{\mathcal{X}}$TEX – ein Überblick

David Kastrup, Markus Kohm, Torsten Krüger,
Michael Niedermair, Rolf Niepraschk

Im Dezember 2002 fand sich eine kleine Gruppe von Ideenträgern und Entwicklern zusammen, um basierend auf $\mathcal{NTS}$ eine Weiterentwicklung von TEX auf den Weg zu bringen. Am Anfang standen dabei einige kaum in Worte gefasste Ideen und die Notwendigkeit, sich in vorhandene Quellen und TEX-Erweiterungen einzuarbeiten. Bereits bevor die Gruppe vom 3. bis 5. Oktober 2003 erstmalig zu einer Klausurtagung zusammenfand, stand nach vielen Experimenten und Begutachtungen fest, dass hochgesteckte Ziele nur zu erreichen sind, wenn große Teile von TEX und damit von $\mathcal{NTS}$ ersetzt werden. Daraus ergab sich der Beschluss, auf Basis der Erfahrungen von $\mathcal{NTS}$, ε-TEX, pdfTEX und Ω (Omega) ein in großen Teilen neues Java-System zu entwickeln – $\varepsilon_{\mathcal{X}}$TEX. Im Folgenden wird der aktuelle Stand der Arbeiten und der Planung wiedergegeben. Gleichzeitig bitten wir um zusätzliche Anregungen und Diskussion des Konzepts.

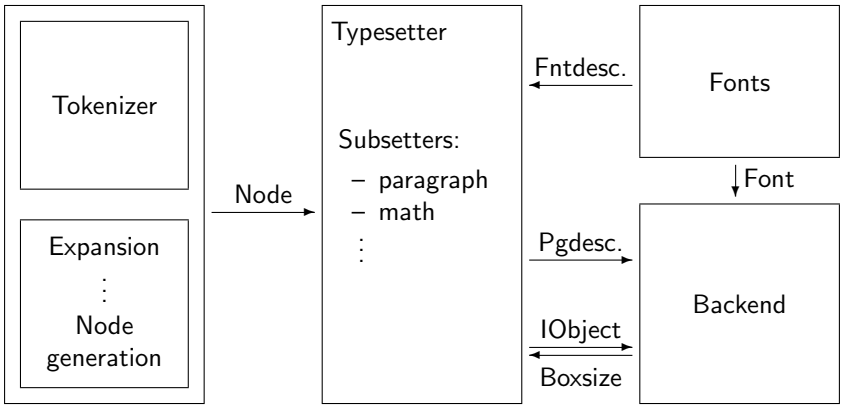

Abbildung 1: Primäres Komponenten-Design von $\varepsilon\chi$TEX

Komponenten

Das objektorientierte $\varepsilon\chi$TEX-Design (siehe Abbildung 1) sieht einige Hauptkomponenten vor, die ohne Wissen über die Arbeitsweise der anderen Komponenten agieren (Kapselung). Auf diese Weise sind das Frontend mit der Tokenisierung und Expansion der Tokens bis hin zur Erzeugung der Nodes getrennt vom Typesetter austauschbar. Gleiches gilt für das Backend mit der Ausgabe-Erzeugung. Eine Sonderrolle nimmt die Font-Komponente ein, die sowohl dem Typesetter als auch der Ausgabekomponente Font-Informationen liefert.

Innerhalb der Hauptkomponenten werden weitere Unterteilungen vorgenommen, die es ermöglichen, einzelne Aspekte der Funktion zu ersetzen oder zu ergänzen. So kann die Font-Komponente durch Hinzufügen oder Austauschen eines Font-Readers durch neue Formate erweitert werden, ohne dass dies unmittelbare Auswirkungen auf andere Teile des Programms hat. Ebenso können in den Typesetter, der für den Umbruch sowie den Aufbau von Seiten zuständig ist, neue Umbruchalgorithmen integriert werden.

Kompatibilität

$\varepsilon\chi$TEX wird alle Primitiven von TEX enthalten. Da jedoch die Syntax und Funktion einiger Primitive – beispielsweise `\input` – auf modernen Systemen

zu unerwünschten Beschränkungen führten, werden in diesen Fällen ergänzende Primitive entworfen und implementiert. Die alten Möglichkeiten werden deutlich als veraltet gekennzeichnet. In einigen Fällen ist dies bereits erfolgt. Schwer nachvollziehbare Eigenheiten von TEX können zu Gunsten einer sauberen Implementierung aufgegeben werden. Ebenso wird das Verhalten im Fehlerfall nicht zu hundert Prozent nachgebildet. Im Endeffekt soll $\varepsilon\chi$TEX in der Lage sein, real existierende Dokumente augenscheinlich identisch zu umbrechen.

Nicht kompatibel wird $\varepsilon\chi$TEX bezüglich der strengen Trennung von iniTEX und virTEX sein. Bisher ist es nur iniTEX möglich, Trennmuster zu laden und Formate zu erzeugen. $\varepsilon\chi$TEX hingegen soll jederzeit Trennmusterdateien global lesen können, also nicht auf die in den Formaten gespeicherte Trenntabellen beschränkt sein. Damit ist es möglich, zur Laufzeit nur die Trenntabellen zu laden, die auch tatsächlich verwendet werden. Auch das Erzeugen einer Formatdatei ist in $\varepsilon\chi$TEX wie in iniTEX in bestimmten Zuständen möglich (im Wesentlichen außerhalb jeglicher Gruppe, also im globalen Kontext).

Von ε-TEX werden im Wesentlichen jene Möglichkeiten übernommen, deren Nutzen sich nicht auf Log-Meldungen beschränkt. Die zusätzlichen Tracing-Primitiven werden nicht in den Sprachumfang von $\varepsilon\chi$TEX übernommen, weil $\varepsilon\chi$TEX für Debugging-Zwecke eigene, weitergehende Mittel zur Verfügung stellen wird. Ebenfalls ausgespart werden die ε-TEX-Möglichkeiten für die Umschaltung der Schreibrichtung. Diese können mit Hilfe von Ω-kompatiblen Primitiven auf Makroebene nachgebildet werden. Damit sollte auch bei real existierenden ε-TEX-Dokumenten Kompatibilität erreicht werden.

Von Ω werden Erweiterungen bezüglich der Schriften sowie die Möglichkeiten zur Beeinflussung der Schreibrichtung übernommen. $\varepsilon\chi$TEX wird jedoch nicht die Encoding-Funktionen von Ω nachbilden, sondern die 16-Bit-Fähigkeit mit eigenen, leicht verständlichen Methoden realisieren. Dabei werden die Encoding-Möglichkeiten von Java genutzt.

Bei der Übernahme von pdfTEX-Features wird unterschieden zwischen neuen Möglichkeiten, die stark an die PDF-Ausgabe angelehnt sind, und zusätzlichen typografischen Fähigkeiten. Letztere sollen weitgehend übernommen werden. Dabei ergibt sich das Problem, dass bei pdfTEX auch diese Primitiven mit dem Präfix „`pdf`“ versehen sind. In $\varepsilon\chi$TEX werden daher neue Namen und teilweise auch andere Primitiven zur Anwendung kommen. Auch hier können Makros als Bindeglied verwendet werden. In den meisten Fällen, etwa bei LATEX-Paketen wie `graphics` oder `hyperref`, empfiehlt sich jedoch

der Wechsel zu einem neuen Treiber. Dies gilt umso mehr, als stark an das Backend angelehnte Möglichkeiten bei ε_{χ}TEX auf andere Art implementiert werden.

Neues

Obwohl ε_{χ}TEX so entworfen ist, dass neue Möglichkeiten in allen Hauptkomponenten, also auch auf Ebene der Primitiven, durchaus mit vertretbarem Aufwand hinzugefügt werden können, soll ε_{χ}TEX bereits von Anfang an mit einigen Neuerungen versehen werden. Dazu gehört die Handhabung unterschiedlicher Kodierungen der Eingabedateien. Dabei stützt sich ε_{χ}TEX auf die in Java eingebauten Möglichkeiten, um beim Lesen von Dateien die Zeichen in die interne 32-Bit-Kodierung zu übertragen.

Da TEX bei diversen Primitiven eine Beschränkung auf 8 Bit und Ω eine auf 16 Bit aufweist, werden dazu entsprechende Äquivalente mit 32 Bit definiert. Dies betrifft beispielsweise `\mathcode`. Bei `\char` hingegen ist eine direkte Verwendung von Werten größer 255 möglich. Zwar ist dies kein zu TEX kompatibles Verhalten. Da TEX jedoch in diesem Fall einen Fehler melden würde, ist das Verhalten zu real existierenden, fehlerfreien Dokumenten kompatibel.

Für Erweiterungen, die nicht vom Ausgabeformat abhängen, sondern bei denen erwartet wird, dass sie von vielen Ausgabeformaten – auch auf unterschiedliche Art und Weise – unterstützt werden können, werden eigene Primitiven verwendet. Beispiele hierfür sind Anfang und Ende von Links und deren Ziel.

TEX kennt Elemente, die nichts mit der Typografie der Seite zu tun haben, aber bei der Ausgabe der Seite (`\shipout`) abgearbeitet werden müssen. Beispiele für solche *whatsits* sind `\openout`, `\closeout` und `\write` jeweils ohne vorangestelltes `\immediate`. Gerade für die genannten Befehle ist eine Auswirkung auf Formatierung und Umbruch der Seite praktisch nie wünschenswert. Leider ist dies jedoch in TEX keineswegs ausgeschlossen. Dieser Zustand soll bei ε_{χ}TEX verbessert werden, obwohl dadurch Inkompatibilitäten entstehen.

Eine Sonderrolle bei den *whatsits* nimmt das Primitiv `\special` ein. Je nach seiner Bedeutung sind Auswirkungen auf den Umbruch erwünscht (etwa wenn es eine Grafik einfügt) oder auch nicht (etwa wenn damit eine Farbe umgeschaltet wird). Davon abgesehen, dass ε_{χ}TEX Farben unmittelbar unterstützen soll, wird es bei ε_{χ}TEX hier ein neues Primitiv für ausgabeformatab-

hängige Erweiterungen geben. Bei diesem Primitiv wird der Typesetter die Entscheidung, ob das Element umbruchrelevant ist und welche Größe es gegebenenfalls haben wird, vom Backend fordern (siehe IObject und Boxsize in Abbildung 1). So kann ein PDF-Backend beispielsweise die Größe einer über dieses Primitiv eingebundenen Grafik zurückliefern.

Mittelfristig soll ε_{χ}TEX auch grafische Elemente wie Linien beliebiger Steigung, Elipsen, Bezierkurven usw. bieten. Ebenso sollen grafische Transformationen wie Rotation, Spiegelung, Skalierung realisiert werden.

Bisher Implementiertes

Tokenizer: Der Tokenizer erzeugt aus diversen Quellen (zum Beispiel einer Datei) einzelne Token und stellt diese in einem Stack zur Verfügung. Dabei wird das jeweilige Encoding der Datei berücksichtigt, welches auch über ein Primitiv in der Datei umgeschaltet werden kann. Es werden dabei alle Standard-Encodings von Java unterstützt (zusätzlich besteht die Möglichkeit, eigene Encodings für bestimmte Fälle zu definieren).

Sonder- bzw. Unicode-Zeichen können dabei auf verschiedene Weise eingelesen wesen:

- Standard-TEX-Variante: `^^`...
- Unicode: `^^^^`*`Unicodename`*`;`
- Omega-Variante: `^^^^^^`*`Hexzahl`*

Wahlweise kann der TokenizerStack in einem Fenster angezeigt werden (visueller Debugger), um das Arbeiten und Erzeugen von Tokens zu verfolgen.

Filemanager: Alle Dateien werden über einen zentralen FileManager geladen. Dabei werden alle Zugriffe in einem Cache gespeichert. Der FileManager sucht und lädt Dateien aus Verzeichnissen, die über `TEXINPUTS` definiert worden sind und zusätzlich aus angegebenen JAR-Archiven.

Dateien, die erzeugt werden, werden im Verzeichnis `TEXOUTPUTS` oder im aktuellen Verzeichnis geschrieben. Ist dieses nicht schreibbar, so wird das Ersatzverzeichnis von `TEXMFOUTPUT` verwendet.

EqTable: Die EqTable dient dazu, alle Primitive, Parameter ... aufzunehmen und diese in den entsprechenden Gruppen-Leveln zu speichern.

Wahlweise kann die Tabelle (Schlüssel und zugehörige Werte) in einem Fenster angezeigt werden (visueller Debugger), um Werte usw. zu betrachten. Dabei steht ein Filter zur Verfügung, mit dem eine Auswahl über den sichtbaren Bereich festgelegt wird. Ein Breakpoint-Mechanismus erlaubt das Anhalten der Verarbeitung, wenn ein Eintrag seinen Wert ändert.

Log-Ausgaben: Ausgaben werden entsprechend einem eingestellten Filter (getrennt nach Terminal und Log-Datei) erzeugt. Dabei kann jeweils ein entsprechendes Encoding verwendet werden.

Register: Bei den Registern wird keine direkte Größenbeschränkung vorgegeben. Diese ist nur vom Zähler (es wird eine 32-Bit-Integer-Zahl verwendet) und vom vorhandenen Hauptspeicher abhängig.

Primitive: Wird schrittweise bearbeitet.

Ungelöstes

Ein Großteil dessen, was in $\varepsilon_{\mathcal{X}}$TEX programmiert wird, soll dieselbe Funktionalität bieten, wie sie bereits in TEX vorhanden ist. Der Weg dahin ist weitgehend vorgezeichnet, wenn auch nicht immer einfach, da es gilt, eine möglichst modulare Struktur zu erreichen. Die von ε-TEX gebotenen Erweiterungen werden voraussichtlich in ähnlicher Weise zu realisieren sein, wie die bereits implementierten TEX-Primitive und Operatoren. Anders sieht es bei den folgenden, bisher nur angedachten Erweiterungen bzw. Änderungen aus. Sie erfordern eine sehr intensive Auseinandersetzung mit der Materie, weshalb die Vergabe von Diplom- oder Doktorarbeiten ideal wäre:

1. Satz von Absätzen, die von der üblichen rechteckigen Form abweichen. Es wird eine generelle Lösung angestrebt, die über das hinausgeht, was das `\parshape`-Primitiv bietet. Es deuten sich etliche Schwierigkeiten an, wenn man mehrfache Umbruchversuche solcher Absätze sowie Seitenumbrüche in Betracht zieht.

2. Schriftenmanagement. Die derzeitige Art, wie TEX Schriften verwendet und verwaltet, weist viele Beschränkungen auf und ist daher sehr umständlich. Die interne Verwendung 32-Bit-kodierter Zeichen innerhalb von $\varepsilon_{\mathcal{X}}$TEX verschärft die Problematik weiter. In Zusammenhang damit stehen auch die folgenden hier nur kurz angedeuteten Problematiken:

   - Mikrojustierung der Buchstabenabstände und -größen

- Optischer Randausgleich
- Sperrsatz u. ä.
- Rechts-Links-Satz

3. Die Möglichkeit der Optimierung der Absatz- und Seitenumbrüche über mehrere Seiten hinweg. Es wäre dazu erstrebenswert, wenn der Seitenumbruch für bestimmte Seiten innerhalb eines $\varepsilon\chi$TEX-Laufs mehrfach erfolgen könnte und ungünstige Ergebnisse verworfen werden könnten. Abgesehen von technischen Fragen für diese Art der Optimierung ist die grundsätzliche Strategie noch weitgehend unbekannt. In Zusammenhang damit steht auch die Möglichkeit, statt eines Gesamtdokuments nur eine Seite oder auch nur Teile davon zu bearbeiten, was ein WYSIWYG-TEX ermöglichen bzw. vereinfachen würde.
4. Registerhaltigkeit der auf der Seite angeordneten Boxen. Damit ist gemeint, dass Textzeilen auf einer Seite nur an bestimmten vertikalen Positionen angeordnet sein dürfen, damit benachbarte Seiten stärker harmonisieren. In Anbetracht dessen, dass auf der Seite nicht nur Textzeilen gleicher Buchstabengröße angeordnet werden, sondern auch Grafiken, mathematische Formeln usw., ist zu erwarten, dass die Lösung nicht trivial ist.

Aufruf

In den drei Tagen, in denen das $\varepsilon\chi$TEX-Team intensiv über eigene und mögliche Erwartungen an ein neues Satzprogramm diskutiert hat, wurde auch deutlich, dass wir noch längst nicht alle Aspekte abschließend beurteilen können. In einigen Bereichen fehlt uns schlicht noch das notwendige Fachwissen.

Gleichzeitig gibt es sicher noch mehr oder weniger dringende Wünsche insbesondere von Format- und Paketentwicklern, an die wir bisher in keiner Weise gedacht haben. Wir hoffen, dass dieser Artikel dazu beiträgt, dass solche Wünsche auch an uns herangetragen werden. Weitere Artikel zu Einzelaspekten von $\varepsilon\chi$TEX werden möglicherweise folgen. Es wäre jedoch schön, wenn wichtige Probleme oder Ideen bereits frühzeitig – also vor der Ausarbeitung und Implementierung grundlegender Funktionen – an uns herangetragen würden.

Wie in Abschnitt „Ungelöstes“ bereits ausgeführt, gibt es auch noch Bereiche, in denen neue Konzepte zu erarbeiten sind. Auch dabei, wie auch bei der reinen Implementierung, ist Unterstützung willkommen und erwünscht.

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

Michael Niedermair
Nusselstr. 2
81245 München
m.g.n@gmx.de

Rolf Niepraschk
Persiusstr. 12
10245 Berlin
niepraschk@ptb.de

Torsten Krüger
Albertinenstraße 20–23
13086 Berlin
torsten@kryger.de

Markus Kohm
Fichtenstraße 63
68535 Edingen-Neckarhausen
kohm@gmx.de

David Kastrup
Kriemhildstr. 15
44793 Bochum
dak@gnu.org