



# Projektvorstellung, Möglichkeiten, Ideen, ...

Michael Niedermair [m.g.n@gmx.de](mailto:m.g.n@gmx.de)

Markus Kohm [markus.kohm@gmx.de](mailto:markus.kohm@gmx.de)

Dante-Tagung, Darmstadt März 2004

## 1 Projektvorstellung

### 1.1 Wie alles begann

- Dezember 2002: Rolf Niepraschk und Markus Kohm suchen nach Möglichkeiten, Java in C++ umzuwandeln (konkret NTS). Einstieg von Michael Niedermair.
- Einarbeitung in den NTS-Code und Tests, ob dieser mit GCJ (GNU-Compiler für Java) in Nativecode umgewandelt werden kann und dadurch schneller wird.
- Januar 2003: Umstellung des NTS-Code und Einbau von Erweiterungen (z. B. Funktionen von  $\epsilon\text{T}_{\text{E}}\text{X}$ , neue und erweiterte Register, ...)

- Sommer 2003: Da der Umbau des NTS-Codes sich als sehr aufwändig herausstellte, wurde beschlossen, ein komplett neues System auf Basis der Erfahrungen von NTS,  $\epsilon\text{T}_{\text{E}}\text{X}$ ,  $\text{pdfT}_{\text{E}}\text{X}$  und  $\Omega$  (Omega) zu entwickeln –  $\epsilon\chi\text{T}_{\text{E}}\text{X}$ .
- Oktober 2003: Klausurtagung in Heidelberg, um Ideen und Möglichkeiten zu sammeln.

## 1.2 Wer sind wir?

Inzwischen ist der Kreis der  $\epsilon\chi\text{T}_{\text{E}}\text{X}$ -Interessierten stark angewachsen. Diese teilen sich auf in

- aktive Programmierer,
- Tester,
- Dokumentationsersteller,
- Berater und
- Interessierte.

Auf der Mailingliste finden sich:<sup>a</sup>

Torsten Bronger

Gerd Neugebauer

Christian Faulhammer

Michael Niedermair

Patrick Gundlach

Rolf Niepraschk

David Kastrup

Heiko Oberdiek

Markus Kohm

Bernd Raichle

Alexander Kraenzlein

Walter Schmidt

Torsten Krueger

---

<sup>a</sup>in alphabetischer Reihenfolge

### 1.3 Konzept



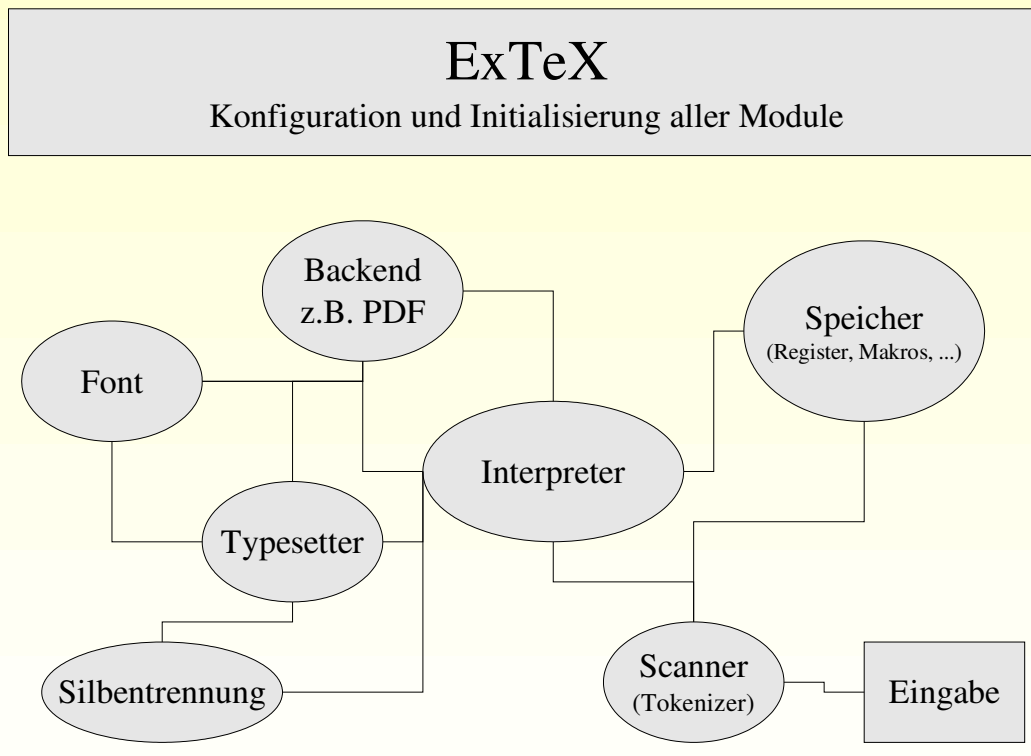
Bitte überlegen, wo man evtl. Mithelfen kann!



- Aufteilung in möglichst unabhängige Hauptmodule, die beliebig ausgetauscht werden können
- Beliebige Konfiguration über Konfigurationsdatei und Aufrufparameter, zum Beispiel Auswahl des Typesetters oder des Silbentrennungsalgorithmus

- $\epsilon\chi\text{T}_{\text{E}}\text{X}$  wird alle Primitiven von  $\text{T}_{\text{E}}\text{X}$  und alle satztechnischen Primitiven von  $\epsilon\text{T}_{\text{E}}\text{X}$  enthalten.
- $\epsilon\text{T}_{\text{E}}\text{X}$ -Primitiven, die zur Fehlersuche dienen, werden durch umfangreiche Debugmöglichkeiten von  $\epsilon\chi\text{T}_{\text{E}}\text{X}$  ersetzt.
- Rechts-Links-Primitiven werden durch die Möglichkeiten von  $\Omega$  realisiert.

- Zusätzlich weitere Primitiven, die die Funktionen erweitern bzw. ausbauen
- entsprechende Formatdateien, so dass  $\epsilon\chi\text{T}_{\text{E}}\text{X}$  Tests von  $\epsilon\text{T}_{\text{E}}\text{X}$  oder  $\text{pdfT}_{\text{E}}\text{X}$  besteht
- Ziel:  $\epsilon\chi\text{T}_{\text{E}}\text{X}$  soll in der Lage sein, real existierende Dokumente augenscheinlich identisch zu umbrechen
- und irgendwann Umsetzung von vielen Ideen, ...  
später mehr dazu



## 2 Module

### 2.1 ExTeX: Hauptbereich

- Einlesen und Auswerten der Konfigurationsdatei (XML-Format)

```
...
<define name="count" class="de.dante.extex.interpreter.primitives.register.
  NumberedCount"/>
<define name="countdef" class="de.dante.extex.interpreter.primitives.register.
  CountDef"/>
...
<Typesetter class="de.dante.extex.typesetter.impl.TypesetterImpl">
  <LineBreaker name="default" class="de.dante.extex.typesetter.impl.
    LineBreakerImpl"/>
  <LineBreaker name="XXX" class="de.dante.extex.typesetter.impl.
    LineBreakerXXXImpl"/>
</Typesetter>
```

- Definition aller Primitiven
- Auswerten der Aufrufparameter
- Initialisierung der gesamten Module
- Start des Interpreters

## 2.2 Scanner

- Einlesen der Zeichen aus einer Datei und Umwandlung in Tokens
- Intern werden die Zeichen im 32-Bit-Format (UTF-32) gespeichert.

- Steuerung über

```
\inputencoding{ISO8859-1}
```

und

```
\inputfileencoding{ISO8859-1}{testnew}
```

- bzw. über Unicodename

```
^^^LATIN CAPITAL LETTER A WITH DIAERESIS;
```

- Für das Encoding werden die Standardencoder von Java verwendet, die sich auch beliebig erweitern lassen.

Big5, Big5-HKSCS, EUC-JP, EUC-KR, GB18030, GBK, ISO-2022-JP, ISO-2022-KR, ISO-8859-1, ISO-8859-13, ISO-8859-15, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5, ISO-8859-6, ISO-8859-7, ISO-8859-8, ISO-8859-9, JIS\_X0201, JIS\_X0212-1990, KOI8-R, Shift\_JIS, TIS-620, US-ASCII, UTF-16, UTF-16BE, UTF-16LE, UTF-8, windows-1250, windows-1251, windows-1252, windows-1253, windows-1254, windows-1255, windows-1256, windows-1257, windows-1258, windows-31j, x-EUC-CN, x-euc-jp-linux, x-EUC-TW, x-ISCII91, x-JIS0208, x-Johab, x-MS950-HKSCS, x-mswin-936, x-windows-949, x-windows-950

## 2.3 Interpreter

- Token: Ausführen, Expandieren bzw. dem Typesetter übergeben.
- ...

## 2.4 Speicher

- Speichern von Registern, Makros, ...
- ...

## 2.5 Typesetter

- Bilden von vertical und horizontal lists
- Zeilenumbruch, Silbentrennung, Absatzumbruch
- Möglichkeit des Umschaltens des Zeilenumbruch- bzw. Absatzumbruchmechanismus (für jeden Absatz)
- Einlesen der Maße für die entsprechenden Glyphen (Höhe, Tiefe, Breite, Kerning, ...)
- Ligaturen
- ...



## 2.6 Font

- Einlesen der Fontmetriken aus
  - TFM (TeX Font Metrik)
  - AFM (Adobe Font Metrik)
  - TTF (TrueType Font)
  - OTF (OpenType Font) in Arbeit
  - OFM (Omega Font Metrik) in Planung
  - ...
- Bilden von Metrik-, Ligatur- und Kerning-Tabellen, etc. für eine allgemeine interne Darstellung

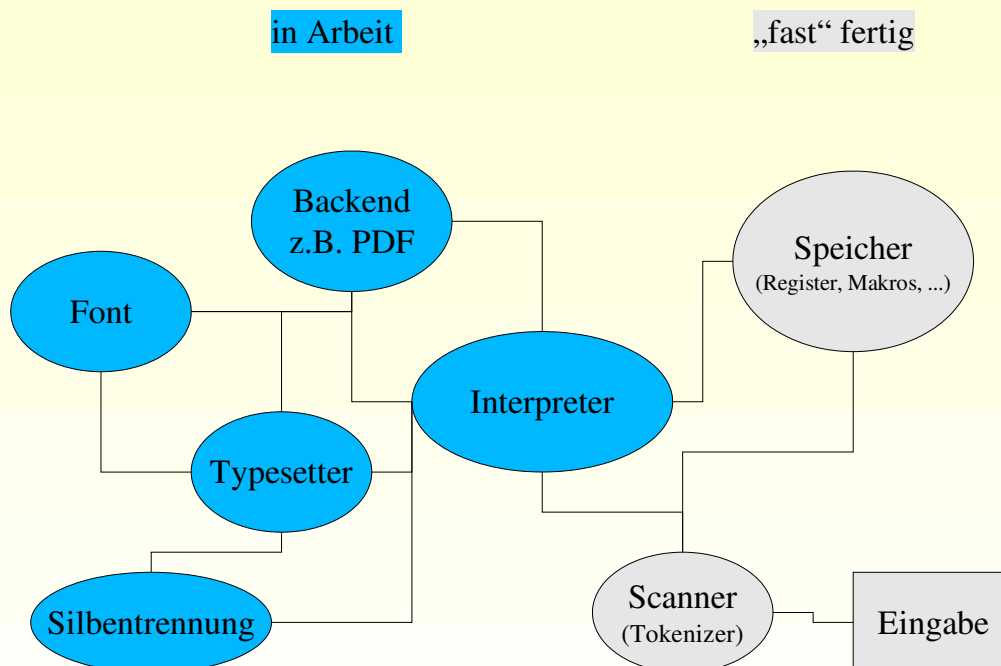
## Ideen

- Optischer Randausgleich: für jeden Glyphen wird angegeben, wieviel rechts und links weggenommen werden kann.
- Kerningdaten in Gruppen (siehe OTF)
- SVG-Fonts
- ...

## 2.7 Backend

- Erzeugen des Output-Formates
- Vorerst drei Formate:
  - PDF
  - Text (erstmal nur für den schnellen Test)
  - Dump (für Debug-Zwecke: hier werden alle Nodes mit Ihren Daten in eine ASCII-Datei geschrieben)

## 3 Stand der Entwicklung



## 4 Was ist geplant?

- $\epsilon\text{T}_{\text{E}}\text{X}$  Funktionen (ohne Umschalten der Schreibrichtung; dieses kann mit  $\Omega$ -kompatiblen Primitiven auf Makroebene nachgebildet werden)
- $\Omega$  (Omega) Funktionen: Möglichkeiten der Beeinflussung der Schreibrichtung

- $\text{pdfT}_{\text{E}}\text{X}$  Funktionen bezüglich typografischen Fähigkeiten (z.B. optischer Randausgleich).  
Primitiven bzgl. Verarbeitung von Bildern (jedoch ohne das Präfix „pdf“) mit neuen Treibern für *graphics* bzw. *hyperref*.  
Unterstützung von zusätzlichen Bildformaten.
- Grafische Elemente wie Linien beliebiger Steigung, Ellipsen, Bezier-Kurven usw.  
Ebenso sollen grafische Transformationen wie Rotation, Spiegelung, Skalierung realisiert werden.

- Einbinden von „anderen“ Eingabeformaten
  - XML (für Daten und Text)
  - XML-FO
  - OpenOffice-Dokumente
  - ...
- weitere Backends
  - SVG
  - ...

## 4.1 Ungelöstes

- Satz von Absätzen, die von der üblichen rechteckigen Form (Höhe, Tiefe, Breite) abweichen.  
Idealerweise an jeder beschreibbaren Figur.
- Schriftenmanagement
  - Mikrojustierung der Buchstabenabstände und -größen
  - Randausgleich (andere, evtl. bessere Methoden)
  - Laufweitenausgleich u. ä. (z.B. durch Spacing, Größenänderung, ...)

- Die Möglichkeit der Optimierung der Absatz- und Seitenumbrüche über mehrere Seiten hinweg.
- Registerhaltigkeit der auf der Seite angeordneten Boxen.
- ...

## 5 Aufruf

- In einigen Bereichen fehlt uns schlicht noch das notwendige Fachwissen. Wer hilft?
- Was haben wir vergessen?
- Wer hat Wünsche?
- Wer hilft bei der Implementierung und beim Ausprobieren neuer Methoden?

# Fragen?