Taco Hoekwater
Siep Kroonenberg
maps@ntg.nl

Redactioneel

Wat later en wat dunner dan de bedoeling was, ligt er nu toch weer een nieuwe MAPS.

Natuurlijk beginnen we weer met de min of meer vaste onderdelen zoals de agenda, het overzicht van de mailing lijsten en de sectie 'Gezeefd van tex-nl'. In de herkansing en vanaf nu een blijvertje is de lijst met andere TEX Gebruikersgroepen.

De MAPS op CD-ROM zou uitgegeven moeten zijn tegen de tijd dat jullie dit lezen. De interface van de CD is zelf ook een TEX bestand, gemaakt door Hans Hagen met ConTEXt uit onze XML database; een verslag hierover kunnen jullie in deze MAPS lezen.

Verder hebben we in deze MAPS: iets over DTP met TEX (er kan dus toch wel wat); literate programming van logo's (let eens op de conclusie van dat artikel); een aantal artikelen over tools in en om TEX (reguliere expressies, Ghostscript, het gebruik van ASCII editors); iets over fonts (wiskundige fonts en het gebuik van de PTT coderingen); natuurlijk weer één en ander over ConTEXt; de nieuwe 4TEX; een verslag van de TUG meeting in Vancouver.

Toen dit nummer geproduceerd moest worden kwamen daar 'ineens totaal onverwacht' allerlei andere dringende zaken zoals conferentie-bezoek tussen. Een tweemans redactie bleek toen wel een erg smalle basis. Maar ineens kwam een heel stel mensen meehelpen. Dankzij al die hulp konden we ook een proeflees-ronde inlassen. We hopen dat jullie dat afzien aan de kwaliteit.

Zo hebben we nu een functionele redactie van meer dan een half dozijn mensen. In onze nieuwe samenstelling wordt er aktief naar artikelen gezocht en worden auteurs aan ons herinnerd door Sven Bovin en Maarten Gelderman. Er wordt proefgelezen door Michael Guravage en Roland Kwee. Het zetwerk en het aanmaken van het print PDF bestand wordt gedaan door ons beiden. Tevens wordt er (zoals in TEX kringen gebruikelijk) een hoop werk verzet door de auteurs zelf, waarvoor ook deze keer weer hulde.

Dat deze MAPS wat dunnetjes is uitgevallen is jammer. Maar gelukkig, ook op het gebied van het verwerven van artikelen hebben we ondersteunende mensen erbij gekregen. Die extra pagina's houden jullie gewoon te goed voor MAPS 24.

Colofon

De MAPS wordt gezet met gebruikmaking van een LATEX class file en een ConTEXt module.

De gebruikte fonts zijn Adobe Times-Roman+Old Style Figures en Frutiger, met een versmalde Courier voor de 'verbatim' omgevingen. Wiskundige formules worden gezet met de MathTime fonts van Y&Y.

Compilatie gebeurt met web2c versie 7.2 onder Linux, en we drukken van een PDF bestand dat wordt gemaakt met pdftex 0.14c en dvips versie 5.84 (gedistilleerd met Adobe's Distiller 3.02), op 90-grams coated papier.

Het versturen gebeurt deels met de PTT partijpost, deels door de internationale bezorging van Kluwer Academic Publishers, deels door locale gebruikers en gebruikersgroepen (waarvoor onze hartelijke dank).

ntg

NTG- en TEX Info

Diverse afkortingen

CTAN – Comprehensive TEX Archive Network; sites waar men 'anonymous ftp' kan gebruiken om TEX/LATEX-achtig materiaal te verkrijgen. CTAN is de 'home' voor de officiële versie van LATEX etc. CTAN sites zijn: `ftp.dante.de`, `ftp.tex.ac.uk` en `ftp.cs.tug.org`

AllTEX – TEX, LATEX, etcTEX

ltxiii – LATEX 3.0

4TEX – Het volledige TEX *runtime systeem* voor Windows PC systemen, gebaseerd op fpTEX en 4DOS.

4allTEX – De 4TEX applicatie *plus* alle mogelijke gerelateerde files en utilities, gedistribueerd op CDROM.

AMS – American Mathematical Society

SGML – Standard Generalized Markup Language

NTG/TUG lidmaatschap

Het blijkt soms dat nieuwe NTG/TUG leden na ongeveer een half jaar nog geen TUGboat of TTN van TUG hebben ontvangen. Ondanks dat men een TUG lidmaatschap via NTG aanvraagt, blijkt in bijna alle gevallen het administratie- en verzendprobleem bij TUG zelf te liggen. Mocht na enige maanden tijd geen post van TUG ontvangen worden, dan worden de betreffende NTG/TUG leden dringend verzocht om contakt op te nemen met het secretariaat van de NTG.

TEX kalender

- 11 november: De 24ste bijeenkomst van de NTG. Het thema is "TEX en taal". Technische Universiteit Delft.
- 8–10 maart: DANTE2000, de 22ste bijeenkomst van de duitstalige TEX Gebruikersgroep. Technische Universität Clausthal Zellerfeld, Duitsland
- 29 april – 2 mei: BachoTEX, de jaarlijkse bijeenkomst van GUST, de poolse TEX Gebruikersgroep. Bachotek, Polen.
- juni/juli: De 25ste bijeenkomst van de NTG. Het thema is nog onbekend. Universiteit Groningen.
- 12–18 augustus: TUG2000, de 21ste globale TUG conferentie. "TEX enters a new millennium". Wadham College, Oxford, Engeland.

MAPS 00.1

Sluitingsdata voor het inleveren van artikelen, bijlagen, en/of mededelingen voor de volgende MAPS uitgaven zijn:

15 januari 2000 (MAPS 00.1; #24)

1 juli 2000 (MAPS 00.2; #25)

De voorjaars-MAPS verschijnt op 1 maart, de najaars-MAPS op 1 oktober.
Aanleveren kopij voor de komende MAPS:

- *Bij voorkeur* gebruikmakend van de LATEX2ε class file `maps.cls` of de ConTEXt module `s-map-01`. Beide files zijn via de redactie te verkrijgen en beschikbaar op de fileserver, archive.cs.ruu.nl (ftp-site)
- Daarnaast kunnen bijdragen ingestuurd worden gemaakt met `ltugboat.sty` of `article.sty` / `report.sty`.
- Verder zijn bijdragen vanzelfsprekend ook welkom in *plain*-TEX of zelfs ongeformatteerd.
- Vector plaatjes bij voorkeur als (Encapsulated) PostScript file plus het oorspronkelijke formaat; dit laatste om eventuele problemen beter te kunnen oplossen.
- Bitmap plaatjes bij voorkeur als PNG (Portable Network Graphics).

Daar MAPS bijdragen in *plain* TEX altijd worden omgezet naar LATEX of ConTEXt, verdient vanzelfsprekend aanbieding van materiaal in een van deze formaten de voorkeur.

Eventuele nadere richtlijnen voor auteurs zijn op te vragen bij de redactie.
Bijdragen kunnen gestuurd worden naar:

Taco Hoekwater,
Singel 191
3311 PD Dordrecht
Email: `taco.hoekwater@wkap.nl`

Personen met een modem maar zonder internetaansluiting kunnen hun bijdrage ook via modem/PTT lijn naar de redactie sturen. Gaarne hiervoor eerst kontakt opnemen met Taco Hoekwater, tel. 078–6137806.

internet

De NTG en het Internet

Jules van Weerden
email: Jules.vanWeerden@let.uu.nl
mmv Maarten Gelderman
email: mgelderman@bik.econ.vu.nl

abstract

De NTG is niet alleen met webpagina's op het Internet aanwezig, maar ook middels een aantal discussielijsten. In dit artikel wordt kort aangegeven wat een discussielijst is, hoe je je aan- en afmeldt en wat je te wachten staat na aanmelding. Tevens wordt een overzicht gegeven van de in Nederland aanwezige TEX-gerelateerde lijsten.

keywords

discussielijsten, tex-nl

Het bekendste onderdeel van het Internet is ongetwijfeld het World Wide Web. In deze bijdrage wil ik het hebben over discussielijsten op het Internet. Eerst zal ik globaal een beeld schetsen van wat zo'n lijst is en hoe een discussielijst werkt. Vervolgens kom ik toe aan het bespreken van de Nederlandse TEX-gerelateerde discussielijsten.

Discussielijsten

Veel mensen zijn zich niet bewust van het feit dat er discussielijsten bestaan. De reden hiervoor is eenvoudig. In tegenstelling tot bij voorbeeld het World Wide Web of ftp heb je voor het gebruik van discussielijsten geen aparte software nodig. Een email-programma (en natuurlijk toegang tot Internet) volstaan. De werking van een discussielijst zelf is eigenlijk te simpel voor woorden. Een discussielijst is voor de gebruiker niets anders dan een email-adres, bij voorbeeld discussielijst@lijstencomputer.nl. Alle mail die naar dit email-adres wordt gestuurd wordt automatisch doorgestuurd naar alle leden van de discussielijst. Het email-adres van de belangrijkste lijst in Nederland, tex-nl, is bij voorbeeld tex-nl@nic.surfnet.nl. Alle mail die naar dit adres wordt gestuurd komt bij alle 183 leden aan.[1] Op de lijstcomputer draait een softwarepakket (LISTSERV en MAJORDOMO zijn de populairste pakketten) dat er voor zorgt dat dit automatisch gebeurt. De op deze wijze gecreëerde lijst kan gebruikt worden om over van alles en nog wat te discussiëren. De TEX-lijsten hebben veelal een vraag- en antwoordkarakter. De leden van deze lijsten stellen op de lijst vragen over TEX-gerelateerde problemen en de andere leden (en met name Piet van Oostrum die hiermee terecht een erelidmaatschap van de NTG heeft verdiend) proberen deze vragen te beantwoorden.

Tot zover erg aardig natuurlijk, maar dan moet een lijst wel leden hebben. Ook dit proces is geautomatiseerd. Naast het adres discussielijst@lijstencomputer.nl bestaat er een adres waarmee het programma direct benaderd kan worden, bij voorbeeld listserv@lijstencomputer.nl. Het is niet de bedoeling dat er naar dit adres gewone berichten worden gestuurd, daar kan de software niets mee. In plaats daarvan moet de mail korte commando's bevatten. Je kan bij voorbeeld een mailtje met als inhoud

```
HELP
END
```

versturen naar listserv@nic.surfnet.nl, je krijgt dan een mailtje met uitleg over de beschikbare commando's terug.[2] Om je aan te melden als lid van bij voorbeeld tex-nl, kan je het volgende mailtje versturen naar listserv@nic.surfnet.nl:

```
SUBSCRIBE TEX-NL Voornaam Achternaam
END
```

De LISTSERV-software kan vervolgens drie dingen doen.

1. Een mailtje terugsturen met de mededeling dat je geen lid kunt worden. Bij tex-nl is het uiterst onwaarschijnlijk dat dit gebeurt, maar bij andere lijsten (b.v. de bestuurslijst van de NTG) kan dit voorkomen.
2. Een mailtje terugsturen met de mededeling dat je lid bent geworden van de lijst. In dit mailtje wordt tevens uitgelegd hoe je je weer af kunt melden. Bewaar het!

1. Dit is niet helemaal waar. Indien het mailtje door één van de 183 leden wordt verstuurd, komt het normaal gesproken alleen bij de overige leden aan. Wil je het ook zelf weer terughebben stuur een melding aan het beheersadres (zie verder) met als commando 'repro <lijstnaam>'. Dit is een faculteit van LISTSERV. Bij de MajorDomo lijsten krijgt iedereen het bericht; ook de afzender. Op sommige lijsten is het bovendien alleen aan leden van de lijst of soms zelfs alleen aan een lijstbeheerder toegestaan om mail te verzenden.

2. Helaas zijn de commando's voor LISTSERV en MAJORDOMO niet identiek.

3. Een mailtje terugsturen met de mededeling dat je voor de zekerheid nog even moet bevestigen dat je echt lid wilt worden van de lijst. Dit is met name om te voorkomen dat ongeldige emailadressen aan een discussielijst worden toegevoegd en om te zorgen dat mensen niet ongewenst aan een lijst worden toegevoegd.

Na deze eenvoudige actie ben je lid van tex-nl en kan je mail (met vragen over TEX en LATEX) naar deze lijst versturen. Tevens ontvang je vanzelfsprekend alle mail die anderen naar deze lijst zenden (gemiddeld zo'n 25 mailtjes per week). Om je weer af te melden stuur je een mailtje met het commando `UNSUBSCRIBE TEX-NL` naar de listserver.

Lijsten in Nederland

Ook op het gebied van de electronische berichtenuitwisseling gaat het goed met de NTG. Op dit moment is een zevental lijsten geïnitieerd vanuit Nederland (voor zover ik weet ;-).

tex-nl Algemene discussielijst over TEX in Nederland.
4tex Lijst voor vragen en mededelingen betreffende het 4TeX-systeem van Erik Frambach en Wietse Dol.
ntg-tex-tools Vragen-, antwoorden- en opmerkingenlijst over allerlei programmatuur die gebruikt kan worden om de resultaten van TEX nog beter te maken. Of zelfs om TEX te genereren
ntg-context Discussie lijst over ConTeXt, een parameter-gestuurd TEX macro pakket
ntg-ppchtex Deze lijst gaat over PPCHTeX, een TEX macro pakket dat gebruikt kan worden om chemische structuurformules te zetten
ntg-toekomsttex Discussielijst over de toekomst van TEX

De eerste twee lijsten zijn te gast op de email-server LISTSERV van SURFNET. De lijsten die beginnen met 'ntg-' zijn te gast bij de MajorDomo email-server van de faculteit der Letteren in Utrecht.

Verder zouden nog de lijsten genoemd kunnen worden:

teTeX lijst, waarop gediscussieerd wordt over het totaalpakket dat Thomas Esser heeft samengesteld van de basis-programmatuur die je nodig hebt om TEX op o.a. UNIX te draaien. Is de basis van de TEX -live CD.
CTAN-Ann De beheerder(s) van CTAN gebruiken deze lijst voor de aankondigingen van nieuwe bestanden op CTAN. Handig om op de hoogte te blijven van het uitkomen van de nieuwste macro's en programma's. Je kunt er zelf (dus) niet posten.

Op alle lijsten wordt levendig gediscussieerd. Voor de lijsten ctan-ann, tex-nl, 4TEX en teTeX houd ik (JvW) zelf een archief bij en omdat het niet erg geheim is, kan iedereen daar ook bij via de ftp-server ftp.let.uu.nl in de directory: /pub/tex/<lijst> [lijstnamen in kleine letters].

Let verder op het verschil tussen de LISTSERV- en de MajorDomo-software. Voor de eerste kun je op de regel 'subscribe <lijstnaam>' ook nog een willekeurige tekst toevoegen. Bv

```
subscribe tex-nl  Jules van Weerden, CIM, Fac \
                                  der Letteren, UU, NL
```

De vrije tekst wordt als kommentaar in de verzendlijst opgenomen en door de programmatuur genegeerd. Bij MajorDomo mag die extra informatie er NIET staan. Die wordt nl beschouwd als het email adres waar de berichten heen moeten.

Als je je voor een lijst wil aanmelden moet je een emailbericht sturen aan het bijbehorende adres. Het onderwerp is niet echt van belang, maar je krijgt het in de reactie terug, dus je kunt het er aan herkennen. In de tekst van het bericht neem je de regel op: subscribe <lijst>.

De verschillende aanmeld-adressen:

- Voor tex-nl en 4TEX: LISTSERV@nic.surfnet.nl.
- Voor ntg-tex-tools, ntg-ppchtex, ntg-context en ntg-toekomsttex: Majordomo@ntg.nl.
- voor teTeX: Majordomo@informatik.uni-hannover.de
- voor CTAN-Ann: listserv@urz.uni-heidelberg.de

De reden dat de lijsten met 'ntg-' beginnen is dat we (nog) geen scheiding hebben aangebracht tussen de lijsten van de faculteit der Letteren en de lijsten van de NTG. Hopelijk kunnen we binnenkort alles splitsen. Dan krijg je ook de reactie terug van MajorDomo@ntg.nl en niet zoals nu van MajorDomo@let.uu.nl.

Voor alle lijsten geldt dat als je een probleem met de lijst hebt dat je een bericht kunt sturen aan
owner-<lijst>@<email-server>
waar een 'echt' persoon achter zit, die kan helpen met het oplossen van het probleem.

Geniet van de lijsten en bestrijd de SPAM (ongewenste reclame).

user groups

TEX user groups around the world

Erik Frambach
email: E.H.M.Frambach@eco.rug.nl
Siep Kroonenberg
email: siepo@cybercomm.nl

All over the world TEX users have formed networks of *user groups* on an informal basis. These user groups consist of enthusiastic TEX users who share their problems and solutions with anyone who wants to join the TEX community. They usually communicate through e-mail and often produce printed periodicals with contributions from members.

The periodicals and discussions on e-mail are essential for users who want to be informed about the latest developments. E-mail is important for getting information on, e.g. what is the newest version of program *x*, where do I find a macro for problem *y*, how do I install program *z*. For this purpose, a list of frequently asked questions ('FAQ') is maintained, and is distributed regularly.

Below we have listed all TEX user groups currently known to us. Much of this information comes from the list of user groups on the TUG website (http://www.tug.org).

AsTEX (French-speaking)
Michel Lavaud, President
Association pour la diffusion de logiciels scientifiques lies à TEX
Association AsTEX
BP 6532
45066 Orleans cedex 2
France
tel: 33 2 38 64 09 94
e-mail: astex-admin@univ-orleans.fr
discussion list astex@univ-orleans.fr

CervanTEX (Spanish-speaking)
Grupo de Usuarios de TeX Hispanohablantes
Association formed in February 1999
José Ra Portillo Fernández:
Departamento de Matemática Aplicada I
Escuela Técnica Superior de Arquitecturs
Avenida de la Reina Mercedes, 2
E-41012 Sevilla, Spain
Public mailing list: spanish-tex@eunet.es
(Send subscription requests to this list)
e-mail:josera@gordo.us.es

CSTUG (Czech and Slovak Republics)
Petr Sojka, President
Československé sdružení uživatelu TEXu
CsTUG, c/o FI MU
Botanická 68a
CZ-602 00 Brno
Czech Republic
tel: +420-5-41212352
fax: +420-5-41212568
e-mail: secretary@cstug.cz
WWW page: http://www.cstug.cz/
periodical: *Zpravodaj CSTUG*

CyrTUG (Russia)
Eugenii V. Pankratiev, President
Irina Makhovaya, Executive Director
Associaciia Pol'zovateleĭ Kirillicheskogo TEX'a
Mir Publishers
2, Pervyĭ Rizhskiĭ Pereulok
Moscow 129820
Russia
tel: +7 95 286 0622, 286-1777
fax: +7 95 288 9522
e-mail: cyrtug@cemi.rssi.ru
WWW page: http://www.cemi.rssi.ru/cyrtug/

Dante e.V. (German-speaking)
Thomas Koch, President
Deutschsprachige Anwendervereinigung TEX e.V.
Postfach 101840
D-69008 Heidelberg
Germany
tel: +49 6221 29766
fax: +49 6221 167906
e-mail: dante@dante.de
WWW page: http://www.dante.de/
periodical: *Die TEXnische Komödie*

DK-TUG (Danish)
Thomas Widman, President
DK-TUG
Department of Mathematical Sciences
University of Aarhus
Ny Munkegade Building 530 DK-8000 Aarhus
Denmark
e-mail: dk-tug-bestyrelse@sunsite.auc.dk

Estonian User Group
Enn Saar, Tartu
Astrophysical Observatory, Toravere
EE 2444 Estonia
e-mail: saar@aai.ee

ϵϕτ (Greek speaking)
Greek TEX Friends Group
Apostolos Syropoulos, President
366, 28th October Str.
GR-671 00 Xanthi
Greece
tel: +30 541 28704
e-mail: apostolo@platon.ee.duth.gr
WWW page: http://obelix.ee.duth.gr/eft/

Grupo de Utilizadores de TeX (Portuguese)
No formal user group yet.
For information, contact Pedro Quaresma de Almeida
Email: pedro@mat.uc.pt

GUST (Poland)
Tomasz Przechlewski (President)
Polska Grupa Użytkowników Systemu TEX
Instytut Matematyki Uniwersytetu Gdańskiego
ul. Wita Stwosza 57
80 - 952 Gdańsk
Poland
e-mail: ekotp@univ.gda.pl
WWW page: http://www.gust.org.pl/
periodical: *GUST*

GUTenberg (French-speaking)
Michel Goossens, President
Groupe francophone des Utilisateurs de TEX
Association GUTenberg
c/o Irisa, Campus universitaire de Beaulieu
F-35042 Rennes cedex
France
tel/fax: +33 5 61 52 83 44
e-mail: tresorerie.gutenberg@ens.fr
WWW page: http://www.gutenberg.eu.org/
periodical: *Les Cahiers GUTenberg*

ITALIC (Irish)
No formal user group yet.
Public mailing list: ITALIC-L@irlearn.ucd.ie
(send subscription requests to
listserv@irlearn.ucd.ie)
Peter Flynn
Computer Centre
University College
Cork, Ireland
e-mail: pflynn@www.ucc.ie

Lietovos TeX'o Vartotojø Grupë (Lithuanian TEX Users Group)
Vytas Statulevičius, Chair
Akademijos 4
LT-2600 Vilnius
Lithuania
tel: +370 2 359 609
fax: +370 2 359 804
e-mail: vytass@ktl.mii.lt
WWW page: http://www.vtex.lt/tex/

Nordic TEX User Group (Scandinavian countries)
Dag Langmyhr, Chair
Nordic TEX Users Group
Department of Informatics
PO Box 1080 Blindern
University of Oslo
N-0316 Oslo
Norway
tel: +47 22 85 24 50
fax: +47 22 85 24 01
e-mail: dag@ifi.uio.no
WWW page: http://www.ifi.uio.no/~dag/ntug/

NTG (Dutch-speaking)
Erik Frambach, Chair
Nederlandstalige TEX Gebruikersgroep
Plantage Kerklaan 65/1
1018 CX Amsterdam
The Netherlands
e-mail: ntg@ntg.nl
WWW page: http://www.ntg.nl/
periodical: *MAPS*

TEXCeH (Slovenian TEX User Group)
Vladimir Batagelj
Jadranska 19
SI-61111 Ljubljana
Slovenia
e-mail: Tex.Ceh@fmf.uni-lj.si

Tirant lo TEX (Grup d'usuaris catalanoparlants de TEX)
Gabriel Valiente Feruglio
Technical University of Catalonia
Department of Software
Mòdul C6, Campus Nord
Jordi Girona Salgado, 1-3
E-08034 Barcelona
Catalonia
e-mail: valiente@lsi.upc.es
Mailing list: catala-tex@aligna.cesca.es (send subscription requests to listserv@cesca.es)
WWW page: http:
//www-lsi.upc.es/~valiente/tug-catalan.html

TUG (International user group, LUG for TeX users not covered by some other group)
Mimi Jett, President
TEX Users Group
1466 NW Front Avenue, Suite 3141
Portland, OR 97209
USA
tel: +1 503 223 9994
fax: +1 503 223 3960
e-mail: office@tug.org
WWW page: http://www.tug.org/
periodical: *TUGboat*

TUGIndia (Indian)
Prof. (Dr.) K. S. S. Nambooripad, Chairman
Indian TEX Users Group
Kripa, TC 24/548, Sastha Gardens
Thycaud, Trivandrum 695014
India
tel: +91 471 324341
fax: +91 471 333186 email:
tugindia@river-valley.com
WWW page: http://www.river-valley.com/tug/

TUG-Philippines (Philippines TeX Users Group)
Dr. Felix P. Muga II, President
Mathematics Department
Ateneo de Manila University
Loyola Heights, Quezon City
Philippines
Tel: (63-2) 426 6001 local 2515
Fax: (63-2) 426 6008
e-mail: fpmuga@admu.edu.ph and
fpmuga@philonline.com

UK TUG (United Kingdom)
Philip Taylor, Chairman
UK TEX Users' Group
For information:
Peter Abbott
1 Eymore Close
Selly Oak
Birmingham B29 4LB
England
e-mail: uktug-enquiries@tex.ac.uk
WWW page: http://www.tex.ac.uk/UKTUG/
periodical: *Baskerville*

user groups

TUG '99, Vancouver

Siep Kroonenberg
Kluwer, Dordrecht
siepo@cybercomm.nl

The twentiest meeting of the TeX User Group was held from monday 16 to thursday 19 august in Vancouver, at the University of British Columbia, UBC in short. The NTG was represented by Erik Frambach and me.

UBC is situated on a beautiful, spacious campus with large trees and some spectacular architecture.

Vancouver makes the most of its American Indian heritage. Souvenir shops sell items decorated with Indian patterns, and totem poles, old and new, are popular as decoration. On the UBC campus there is an anthropological museum with a very nice collection of Indian and Eskimo sculpture and other art.

Vancouver is mostly a sprawling city, with much greenery. However, the downtown area, situated on a peninsula, contains a lot of high-rise buildings, mostly new and handsome. There is also an older section of the downtown area, Gastown, with many places for going out.

Going downtown for dinner involved a long and grueling bus ride, with stops at just about each of the many intervening blocks. Unless, of course, you have a car, in which case it still isn't a quick trip.

Monday: 'TeX and Math on the Web'

Stephen Fulling, a math professor who tries to leverage internet for his classes, described his frustration with all the available not-quite-good-enough options for representing math in web pages. Subsequent talks suggested that his frustrations may rapidly diminish in the time to come.

The next few paragraphs are a combined summary of background material presented in those subsequent talks.

SGML, short for Standard Generalized Markup Language, has been in use for a long time by large publishers and by corporations which need to manage large amounts of documentation. SGML is not exactly a file format, but a language to define document formats. HTML is such a format defined in SGML terms.

SGML is meant to define document structure, not appearance. Defining the latter requires a second formalism, DSSSL. In the case of HTML, the browser decides how a document is to be rendered.

HTML brought SGML under the attention of a large public, and now there is an enormous surge in activity around SGML. One development is the emergence of XML (eXtensible Markup Language), which is simpler and a lot more manageable (both for computers and for humans) than SGML, and many people have great hopes for XML as an exchange format for documents.

For formatting, the counterpart of DSSSL for XML is XSL, eXtensible Stylesheet Language. It allows specification of transformations from one XML 'vocabulary' into another, in particular from a content-oriented vocabulary into a presentation-oriented one. Software support for XML and XSL is rapidly growing.

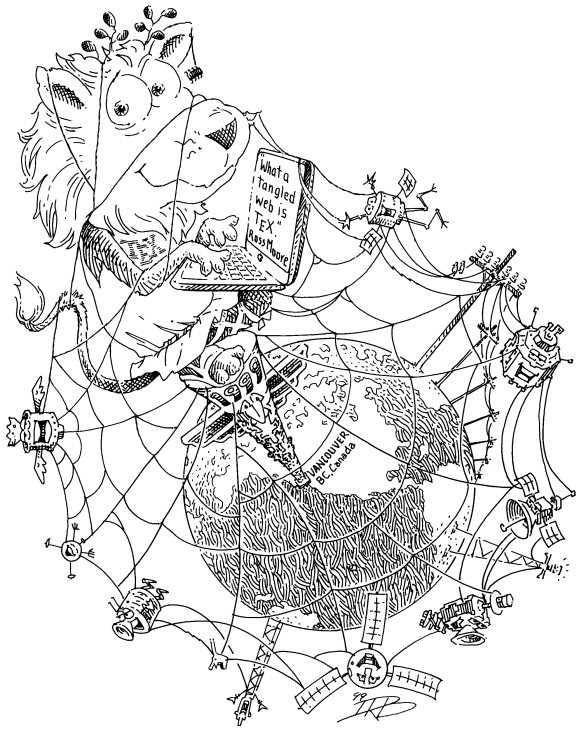
Patrick Ion gave a talk about MathML, Mathematical Markup Language, which should provide math capabilities for web pages. He is a member of the W3 Math Working Group (WG), which is responsible for the MathML specification. This WG contains representatives from a broad section of the information industry, including e.g. IBM, HP, Mathematica, Netscape, Boeing, Microsoft and the AMS. Work on version 2.0 is now underway.

MathML is designed as an XML application, with the familiar angle brackets, and bears no similarity to TeX markup. He showed a MathML content coding and a presentation coding of a very simple equation. What both codings had in common was their extreme verbosity: clearly, MathML is not designed for hand-coding. It is, on the other hand, designed to be read and written by computers. As a consequence, there is a lot of work underway to build MathML support into web browsers, wordprocessors and math software.

Doug Lovell described TeXml, an XML vocabulary for representing TeX source together with a Java program TeXMLatté for translating TeXml into TeX, which can be run as a browser plug-in or as a program on the web server. The rendering of the resulting TeX code can be provided by IBM's techexplorer program, which is another browser plugin.

The next talk, by Paul Topping from Design Science, about generating MathML with the MathType equation editor, unfortunately had to be canceled.

Eitan Gurari and Sebastian Rahtz discussed conversions



from LaTeX to XML and HTML, with special attention to tex4ht. This program requires the use of a stylefile that places specials in the dvi file, which are needed for the conversion process. Gurari also gave a demonstration of different styles of markup and of corresponding XSL transformations, using a cookbook recipe as an example. In the course of this presentation, the coding of the recipe acquired more and more database features.

Subsequently, Chris Rowley gave his amusing views on TeX and XML.

Donald P. Story presented interactive teaching material – including tutorials, technical papers and games – created with the aid of (La)TeX, Acrobat and occasionally JavaScript. Check out his website: `www.math.uakron.edu/~dpstory/acrotex.html`.

The business part of the day was concluded with a panel discussion. Points raised:

- rendering of MathML by web browsers
- inclusion of math equations in web search engines
- a concern that a MathML standard with accompagnying fonts might stifle the development of new mathematical notation
- since obviously XML/MathML is far from ideal for hand-coding, there might be a role for LaTeX as an input format to be converted to XML
- there certainly seems to be a role for TeX as typesetting backend for XML
- it was hoped that there would be adequate coordination between the various people working on conversions between (La)TeX and XML

Tuesday: Customizing Document Layout

Jean-Luc Doumont uses TeX for typesetting all kinds of documents for which TeX is not the obvious choice, such as newsletters and promotional brochures. He uses his own collection of macros. He tells his clients that he uses 'a system he programmed himself', which seems to inspire more confidence than mention of TeX and the name of Donald Knuth. Other independent typesetters at the meeting confirmed that it is unwise to tell clients that you are using TeX. Printers are even willing to blame TeX when they select the wrong Pantone color!

Peter Flynn presented a macro package Vulcan for LaTeX for filling in some obvious omissions from the LaTeX standard classes: better author information, such as addition of entries for affiliation and email, and publication-oriented metadata such as `\journal` and `\vol`. Vulcan also provides improved formatting.

Some vendors introduced new versions of their programs:

Richard Kinch's TrueTeX now supports Omega and Unicode, and the algorithms of the font program MetaFog have been dramatically improved.

Blue Sky's Textures TeX implementation for the Macintosh now also has something to offer as a QuarkXPress plug-in: math equations embedded in regular text, and improved line breaking.

PCTeX now can print PostScript graphics to arbitrary printers.

The LaTeX3 team reported on new features in LaTeX. A new version of LaTeX, possibly to be named LaTex2e*, should be released some time next year. Chris Rowley demonstrated the new template syntax, with a keyword–value syntax. This should become available later this year as a package running under LaTex2e. There are thoughts about making this a replaceable frontend. A replacement candidate would be an XML parser. David Carlisle discussed work underway for more predictable handling of the vertical list, and improved float handling.

Later that day there were workshops on using docstrip by Michael Doob and on converting LaTeX2.09 stylefiles to LaTeX2e classfiles by Anita Hoover.

Wednesday: TeX in publishing

Kaveh Bazargan runs an independent typesetting firm Focal Image, which does a lot of work for large publishers. More and more, publishers require different kinds of output for a single publication: besides printed output, they might require hyperlinked pdf or html. Focal Image makes sure that such output can be generated from the LaTeX they produce.

He had some things to say about what publishers might do to induce their authors to produce good LaTeX. He proposed an author kit, *not* including the actual class file but only a generic version, and also including an authors guide and a copy of Lamport's book. He also proposed an 'Editor's Guide', which would enable copy editors to treat LaTeX-generated text more intelligently.

For new editions of books, it frequently happens that the author uses the Focal Image versions of the electronic files of the previous edition as a starting point.

Frederick Bartlett described his activities for Springer Verlag regarding XML conversions.

Harry Payne had a tale from the trenches about putting together the proceedings of a conference which also had to appear as web pages. Better preparation and better guidelines for authors could have saved him an enormous amount of work.

Conversion to html was done with latex2html, which was the best option at the time. Converting the entire book at one go would take 750 MB of memory and the better part of an afternoon.

He advised to send the manuscript to the publisher only after the web edition was done, as a protection against 'gigantic blind spots'.

He edited papers as stand-alone documents, and used a script to turn them into 'include'-versions. The document as a whole was compiled with a makefile.

Robert Kruse presented a system PreTeX for managing the production of large-scale technical books such as math and science textbooks, textbooks on programming or documentation for large software projects. Such a project often involves the production of additional material, such as answers to exercises or source code. There also may be lots of graphics from various sources.

The file format used is called preTeX. The preTeX preprocessor can generate the various items, including TeX source, from this format. Partial compilation and file management facilities are provided.

In another talk, Paul Mailhot showed how interactive pdf's can be produced with this system. His example was a book on programming with clickable examples of source code.

Art Ogawa discussed the use of TeX for database publishing and the specific requirements and possibilities of such TeX applications.

Hu Wang demonstrated a setup for authors to submit abstracts in LaTeX format via a web page. This setup will validate the LaTeX code, but also provides for authors who don't want to use LaTeX. Inclusion of figures is also possible. This system allows automated typesetting of abstract books.

Peter Sojka described methods for creating custom hyphenation patterns, which can be very useful for productions such as dictionaries.

This day's final talk was on Jonathan Fine's Active TeX (all characters are made active) and DOT input syntax, conceptually similar to SGML. Art Ogawa served as virtual Jonathan Fine. A quote (from Ogawa as Ogawa): 'Not seeing the backslash for long stretches does strange things to your head'.

Points raised during the day's panel discussion:

- Nadia Molozian from Harcourt Publishers noted a strong increase in the use of LaTeX in production at her company. An advantage of LaTeX is that copy editing involves less work.
- Generally, LaTeX submissions by authors also appear to be up, although this is not true everywhere.
- Production of conference proceedings is a messy business; often, quick-and-dirty measures such as photographic resizing must provide a semblance of consistency.
- The publisher has little chance of influencing the coding style of monographies. Often, the author has been working on his book for years before a publisher gets his hands on it.
- An interesting speculation by Frederick Bartlett on why authors like to use bad LaTeX coding: writing is hard work; authors cast about for distraction and find it in fiddling with appearances.
- The same speaker encouraged the audience to complain to publishers about bad-looking books; this would give publishers an incentive to let their TeX specialists do something about it.

In the evening, the results from a poetry contest were pre-

sented. The declamation of poetry required one to dress up as the TeX lion, with a knee-length t-shirt and sprigs of greenery tied to one's head. Some of samples:

Richard Kinch:

A young maid desired to woo:
'Give me TeX', she would heatedly coo
his code became tangled
in her macros new-fangled
in a passion of boxes and glue

Christina Thiele: *Under the volcano*

Those daring young men and their computing machines,
All night long new code they dispute.
All day in their dreams
screen flow extremes
beat the code till it pleads, 'Please reboot!'

Sebastian Rahtz: *A TeX Haiku*

```
\expandafter\def
\csname def\endcsname
{\message{farewell}}\bye
```

The full collection should appear eventually on the tug99 web site: `www.tug.org/tug99`.

Thursday: Fonts, Graphics and New Developments
Jean-Luc Doumont, whose work we had already seen on the second day, now showed us his TeX macros for producing graphics. The results were as handsome as the work he had shown on the previous occasion. He is more than willing to make his macros available if people are interested; you can contact him at `JL@JLConsulting.be`.

Wendy McKay and Ross Moore developed a package WARMreader to add TeX text labels to graphics, using a flexible system for anchoring and formatting those labels. Contrary to psfrag, their system works for arbitrary graphics, since the text labels float on a separate layer. They also created a utility Zephyr running on the Macintosh to interactively add those text labels.

Michael Doob presented AcroDVI from Lesenko and Siebenmann. Since he failed to get the program to run on his own system, we was limited to a verbal presentation. In a fit of silliness, he found occasion to point out an instance of Brouwer's Fixed Point Theorem in a photograph taken at the conference which featured the screen it was projected on.

The next virtual speaker was Arthur Ogawa representing Alan Hoenig. The latter has developed a program MathKit to generate math fonts to combine with any commercial font family. Visual harmony was achieved by tuning the parameters of the Computer Modern MetaFont math fonts. Hoenig has put together a number of parameter sets. In most cases, one of those sets will be appropriate. MathKit is implemented in Perl scripts ('sufficiently idiosyncratic and quirky to make it a good match for TeX') which generate everything needed to use the results in TeX and LaTeX.

The morning session was concluded with demonstrations of the above-mentioned utility Zephyr and of techexplorer. The latter is a browser plug-in running on Windows and Unix platforms for rendering a subset of LaTeX, with some interesting interactive options. We saw a very nice demonstration of a book converted to techexplorer format.

Fabrice Popineau opened the afternoon session with a talk about fpTeX, his Win32 implementation of the standard Web2c TeX. When he started work on this, Christian Schenk had already a version of MikTeX, another free Win32 TeX implementation. However, the advantages of compatibility with the Unix web2c standard were sufficient reason to undertake this port. He described the technical issues that had to be dealt with: what compiler to use, what to replace shell scripts with, how to work around differences in file systems, whether or not to use the registry (the answer: not). The dvi-viewer windvi raised a lot of additional issues. He also gave us a glimpse of his new texconfig tool.

Jeffrey McArthur, who does a lot of database publishing, took us through the planning and setting up of a large TeX-based software project, and had some interesting war stories to tell.

The final speaker was Timothy Murphy, who is working on a web2java project. This project is unrelated to the NTS reimplementation of TeX in Java. He argued that the use of Java created a great opportunity to modularize dvi drivers. Along the way, he did his level best to provoke the audience, by labeling all efforts to extend TeX as schismatic, and claiming that dvi was the greatest format ever for formatted output, but the audience didn't take the bait.

The official meeting ended with a panel discussion on the future of LaTeX. Points raised were:

- LaTeX is broken, people are fixing it in uncoordinated ways (Arthur Ogawa). The volume of the combined macro packages, even only the supported ones, dwarfes that of the LaTeX core.

- more modularity, more hooks (several speakers and members of the audience)
- LaTeX as backend for typesetting XML (Frank Mittelbach)
- A better designer interface (Steve Grathwohl). David Carlisle pointed out that this should be taken care of by the new template interface
- Frustration with floats (several people)
- Frank Mittelbach also pointed out that LaTeX had not exactly been created for professional publishing, and that building better internals would break a lot of packages. Arthur Ogawa pleaded for improving the internals anyway, and argued that package writers would certainly be willing to rewrite their packages to take advantage of the new internals.

Throughout the meeting, we could admire a collection of typographic experiments by Alban Grimm which Frank Mittelbach had brought with him. Alban Grimm, who is professor in Typography at the University of Mainz, used Metafont and text fragments from the Apocalypse to produce some striking calligraphic effects. At the end of the meeting, we could all select some sheets to take home.

The meeting was concluded with a banquet, which offered further opportunity for fraternization.

From this conference I got a definite impression that TeX is once again in an upswing, and that this may have something to do with its suitability as a web authoring tool for technical material and with the quick response of the TeX community to web developments.

tex-nl

Frans Goddijn
frans@iaf.nl

Gezeefd uit de TEX-NL discussielijst

abstract
This is part six in a series of articles started in 1993 by Philippe Vanoverbeke. Philippe selected a number of messages from the TEX-NL mailing list which deserved to be remembered and looked up by readers of Maps. Solutions, hints and tips. Maps of spring 1999 had part 5.

For this installment, a number of messages has been selected that were published on TEX-NL march 1999 through july 1999. As before, honorary ntg member Piet van Oostrum proved to be invaluable with his swift responses. Hans Hagen is also strikingly helpful, on (and off) this list and on the NTG–CONTEXT list.

The filtering method for these articles is haphazard; anything the editor (a simple user) can't grasp doesn't make the cut.

Thanks to *all* active subscribers of TEX-NL!

keywords
TEX-NL, Oostrum, Hagen, guru, tips, hints, hack, bug

Door Erik Frambach <E.H.M.Frambach@ECO.RUG.NL>: **Tip**

Ik weet niet of dit algemeen bekend is, maar via

`http://listserv.surfnet.nl/archives/tex-nl.html`

kun je het hele TEX-NL–archief (teruggaand tot 1995) bekijken en doorzoeken. Erg handig!

Variabel wit tussen koppen **vraag**

Ik zou willen dat wanneer een `\chapter` direct begint met een `\section` (dus zonder tussenliggende tekst) er tussen de chapter–titel en de section–titel minder wit is dan het wit boven een normale section–titel + het wit onder een normale chapter–titel; misschien zelfs helemaal geen wit. Hoe moet dat ook weer?

Door Piet van Oostrum <piet@CS.UU.NL>: **antwoord**

Je kunt met `\unskip` de witruimte onder de hoofdstuk titel weghalen. Met `\vspace{..}` kun je dan de gewenste hoeveelheid toevoegen:

```
\chapter{Test}
\unskip\vspace{..}
\section{Section}
```

Je kunt natuurlijk gaan zitten knutselen met `\section` zodat ie zelf automatisch detecteert dat er geen tekst voorstaat, maar dat lijkt me verspilde moeite.

10 karakters per inch **vraag**

Bij het aanmaken van overschrijvingsformulieren mag je maximaal 10 monospaced karakters per inch zetten (met letterhoogte tussen 2.50mm en 3.20mm). Welke fonts kan je daarvoor gebruiken?

Voor de optische regels moet je OCR-B (die zijn 10 cpi) karakters gebruiken, maar die laten geen accentkarakters toe. En bij het drukken van namen heb je soms accentkarakters nodig.

antwoord Door Taco Hoekwater <taco.hoekwater@WKAP.NL>:
Hoogte is het probleem niet, vrijwel elk typewriter font valt onder die catagorie. CMR `\tt` is prima geschikt. Breedte is even proberen:

```
\tt
\setbox0=\hbox{ABCDEFGHIJ}
\message{Breedte: \the\wd0, Hoogte: \the\ht0 }
```

(1pt=approx 0.351mm)
Het gebrek aan accentkarakters is nou juist de bedoeling. Geeft wat lelijke typografie, maar het komt in ieder geval aan bij de juiste persoon.

antwoord Door Piet van Oostrum <piet@CS.UU.NL>:
cmtt10 is 5.25pt breed, cmtt12 6.175pt. De laatste dus 0.0857 inch dus nog te klein. je zult hem dan dus nog moeten vergroten (`scaled magstep1` bijvoorbeeld). De hoogte van een hoofdletter wordt dan 8.8pt=3.1mm. Voor de staarten komt er dan nog 3.2pt=1.13mm bij, dus als die meetellen zit je net 0.13mm boven de maximale grootte, maar ik kan me voorstellen dat dat binnen de meetfout zit.

vraag **Caption *naast* figuur**
Ik wil graag de caption *naast* de figuur hebben (gewoon omdat daar soms veel ruimte over is). Weet iemand hoe ik dat voor elkaar moet krijgen?

antwoord Door B. G. H. Gorte :

`/usr/share/texmf/doc/latex/sidecap/README:`

The sidecap package. This package defines the new environments SCfigure and SCtable, analogous to figure and table, which make it easy to typeset captions sideways. The package knows the options outercaption, innercaption, leftcaption and rightcaption.

antwoord Door Edwin Gelinck <e.r.m.gelinck@WB.UTWENTE.NL>:
Uit *Using EPS Graphics in LaTeX2e documents* door Keith Reckdahl:

```
\begin{figure}[htbp]
 \centering
  \begin{minipage}[c]{.45\textwidth}
   \centering
   \caption{Caption on the Side}
   \label{fig:side:caption}
 \end{minipage}%
 \begin{minipage}[c]{.45\textwidth}
   \centering
   \includegraphics[width=\textwidth]{box.eps}
 \end{minipage}
\end{figure}
```

vraag **Van PDF naar tekst**
Hoe converteer je van PDF naar tekst?

antwoord Door Maarten Gelderman <mgelderman@BIK.ECON.VU.NL>:
In een aantal gevallen is dit de ideale manier om bestanden naar Word of iets dergelijks te converteren: PDF–bestand opvragen in Acrobat Reader, alles selecteren, knippen en plakken in Word en geen uitgever die mekkert. Aan slechte Word–documenten zijn ze gewend. . .

Door Hans Hagen <pragma@WXS.NL>: **antwoord**
Dat kan veel sneller!

```
pdftotext filename.pdf
```

zit in de web2c distributie (soms als pdftotex). Mekkert soms wat over color spaces, maar dat kun je gewoon negeren. Kan ook met kolommen overweg. Ik ben net bezig in tk een editor te maken waarmee ik zo'n tekst in kolommen kan fatsoeneren, vandaar.

Door Luc De Coninck <luc.de.coninck@KH.KHBO.BE>: **antwoord**
Van een collega kreeg ik volgende link i.v.m. LaTeX en PDF.

```
http://online.redwoods.cc.ca.us/instruct/darnold/StaffDev/Index.htm
```

Prutsen in PostScript–figuur **vraag**
Ik zoek een tool/combinatie van tools om een beetje te prutsen aan een PostScript–figuur voor ik die in LaTeX importeer. Het gaat meer bepaald om een figuur waarop het logo van de software waarmee zij gegenereerd werd op voorkomt. Dat logo zou ik er uit willen voor/tijdens de conversie van .ps naar .eps

Door Siep Kroonenberg <Siep.Kroonenberg@WKAP.NL>: **antwoord**
Soms is het mogelijk om de figuur in een ascii–editor te bewerken en het logo eruit te pikken.

Verder kunnen sommige tekenprogramma's vrij algemene PostScript lezen; als dat lukt kun je het logo in dat tekenprogramma verwijderen. Illustrator 7 kan pakweg 75% van alle PostScript bestanden (eps of niet) die ik binnenkrijg inlezen.

Er zijn ook een aantal freeware tooltjes — die allemaal Ghostscript gebruiken — die PostScript omzetten naar een meer hanteerbare vorm zodat je ze ook kan inlezen in bijvoorbeeld CorelDRAW of MayuraDraw: ps2ai zit bij de Ghostscript distributie in; je kunt de pdfwrite en/of epswrite uitvoerformaten proberen; Boguslav Yackowski heeft ps_conv gemaakt.

Hoe kan ik vertikaal uitlijnen in een tabel? **vraag**

Door Werenfried Spit <W.Spit@WITBO.NL>: **antwoord**
dcolumn.sty doet dat:

```
\usepackage{Dcolumn}
\begin{document}
\begin{tabular}{lD{.}{,}{8}r}
a & 1.0  & een \\
b & 2.02 & twee \\
c & 33.003 & drie \\
d & 444    & vier \\
e & .0000005 & vijf
\end{tabular}
```

Door Sven Bovin <sven@HARTREE.CHEM.KULEUVEN.AC.BE>: **antwoord**
Je kan ook de tabel definiëren met (voor de decimaal uitgelijnde kolom)

```
r@{\hspace{0mm}}l
```

en de decimale punt in de (links uitgelijnde) „decimale kolom" plaatsen. Op die manier verdwijnt de spatie tussen de twee kolommen en kunnen 84 en 0.025 zo worden uitgelijnd dat de 4 en 0 boven elkaar komen.

vraag **Orphins, delphans**

In mijn (book) document komt het verscheidene keren voor dat een paragraaf gesplitst wordt over twee bladzijden met maar een enkele regel op een van de twee bladzijden. Wordt dit niet „orphins" genoemd? Enfin, ik denk dat ik zulke paragrafen liever samen hou op een bladzijde. Is dit via parameters te beïnvloeden/ontraden in LaTeX?

antwoord Door P. Wackers <P.Wackers@LET.KUN.NL>:
Zie de bespreking in Lamport's Guide op p. 191–192 en in de Companion op p. 99–100.

Samengevat: `\pagebreak` in een alinea leidt tot een nieuwe pagina bij regel volgend op de woorden waartussen het commando staat. `\nopagebreak` doet het omgekeerde `\samepage` kan een aantal dingen bij elkaar houden.

Mij bevalt het beste:

```
\type{\enlargethispage{size}}
```

bijvoorbeeld:

```
\type{\enlargethispage{\baselineskip}}
```

waarbij je wel moet bedenken, dat wanneer je meer dan een regel extra ruimte neemt, dat erg opvalt. In veel lastige gevallen is `.5\baselineskip` echter al genoeg om het ongewenste effect te verdrijven.

antwoord Door Gilbert van den Dobbelsteen <gilbert@LOGIN-BV.COM>:
In het engels: widows and orphans. je zou dan verwachten dat de Nederlandse vertaling weduwen en wezen zou zijn, echter in klassieke drukkers–terminologie heten die dingen hoer en hoerenjong.

Maar iedereen snapt natuurlijk wel dat je dat soort kretologie niet in een dialoog boxje kunt zetten van de zoveelste tekstverwerker. Dus dat heet dan zoiets als *Zwevende regels voorkomen*.

Goed, sorry, dat is natuurlijk geen antwoord op je vraag. Ik vermoed dat je wel iets met penalties kunt instellen.

vraag **Beste plaatjes in LaTeX**

Ik studeer scheikunde en heb na ernstige horrorverhalen over Word besloten m'n hoofdvak–verslag in LaTeX te gaan maken. Ik werk met een Linux (Redhat 5.2) systeem met de `tetex` installatie. Ik gebruik LyX als front–end om niet meteen al te zwaar LaTeX in te duiken ;=)

Om mijn AIO tevreden te houden moet ik helaas wel met Excel files werken (die hij eventueel zelf later kan gebruiken voor presentaties en dergelijke), nu moet ik dus de Excel grafieken in LaTeX krijgen. Tot op heden doe ik het volgende:

- Maak nieuwe grafiek in Excel als los object op eigen pagina
- Knip–Plak naar Paint Shop Pro, sla op als PNG
- Onder linux, met convert uit het ImageMagick pakket omzetten naar `eps`
- Gebruiken in LaTeX

De kwaliteit van de figuren is nogal slecht, erg blokkerig waardoor bijvoorbeeld as–bijschriften slecht leesbaar zijn. Ik heb de verschillende PS formaten geprobeerd... ik zie nauwelijks verschil. Mijn vraag is dan ook: zijn er nog tips en truuks om tot een beter resultaat te komen?

antwoord Door Siep Kroonenberg <siepo@CYBERCOMM.NL>:
Op die manier krijg je een bitmapped weergave, op waarschijnlijk veel te lage resolutie.

Je kunt beter een Postscript printerdriver een eps–file laten aanmaken; dit configureer je in de Postscript tab van de printerconfiguratie dialoog. Zorg dan meteen dat naar file wordt „geprint”. Met GSView moet je vaak nog de boundingbox aanpassen.

Het kan nog uitmaken welke Postscript printer je installeert. De driver moet bij voorkeur, en onder NT beslist, van Adobe (www.adobe.com) zijn. Een Postscript driver bestaat uit een algemeen gedeelte en een bestandje met printer–specifieke informatie. Als je meer dan 1 Postscript printer hebt geïnstalleerd zullen die het algemene gedeelte van de driver delen. Kies een generieke driver, bijvoorbeeld de Apple Laserwriter Plus v38.0 die met bepaalde versies van de Adobe driver wordt meegeleverd, of de driver voor Distiller invoer. De laatste heeft de voorkeur, tenzij je nog een PostScript level 1 printer gebruikt.

Door Piet van Oostrum <piet@cs.uu.nl>: **antwoord**

Doe vanuit Excel een Print naar een postscript printer met de print–to–file optie aan. Als je geen postscript printerdriver hebt, installeer er dan een. Als de printerdriver de mogelijkheid heeft kies dan voor een zo printer–onafhankelijk mogelijke instelling (kies voor portabiliteit), of nog beter: eps. Waarschijnlijk is je eps dan nog niet goed genoeg en moet je hem met pstoedit (of onder windows: psview met pstoedit geïnstalleerd de menu entry *convert to editable format*, en dan postscript kiezen) converteren naar fatsoenlijke postscript.

Door Siep Kroonenberg <siepo@CYBERCOMM.NL>: **antwoord**

De PostScript die je van MSOffice krijgt is meestal onherstelbare bagger: Bézierkrommen opgebroken in kleine lijnstukjes die dan ook nog eens losse objecten zijn inplaats van een polygoon; woorden die gefragmenteerd zijn tot losse letters; vlakken die uit arceringen zijn samengesteld. Geen enkele automatische conversie kan dat repareren. Gewoon niet aankomen en hopen dat de printer het slikt.

Bij Kluwer krijgen we ook regelmatig eps–en met lijntjes van dikte 0. Dat levert op een 600dpi desktop–printer geen problemen op, maar die lijntjes hebben de neiging te verdwijnen op een 2400dpi belichter en zien er op een Docutech ook niet jofel uit. Als de PostScript code niet al te onoverzichtelijk is doe ik daar weleens wat aan met een ascii–editor, maar anders wordt het de botte–bijl methode: met Ghostscript er een 600dpi pcx bitmap van maken en die in Photoshop bijsnijden en converteren naar eps; dat levert lekker grote bestanden op maar ziet er wel veel beter uit dan de 72dpi van PSP. Bijsnijden kan eventueel in PSP; voor conversie naar eps kun je eens kijken naar de netpbm utilities, beschikbaar voor Win32 onder CTAN.

De Ghostscipt opties voor conversie naar bitmap zijn:

```
-sDEVICE=pcxgray (of pcxmono of pcx24b) -r600 -sOutputFile=....
```

Door Werenfried Spit <W.Spit@WITBO.NL>: **antwoord**

- vanuit Excel direct postscript afdrukken, dat werkt heel goed. je moet dan wel een Adobe postscriptdriver gebruiken (geen native w95, die maakt slechte PS) en die instellen op encapsulated postscript. de drivers van adobe zijn via hun website te krijgen
- je kunt via Excel ps–tricks of gnuplot input maken en die dan importeren

Door Edwin Drost <edro@OCE.NL>: **antwoord**

Ik heb dit probleem ook een keer gehad en deed het als volgt:

Maak de grafieken als aparte sheet en print naar PostScript (Apple laserwriter Plus driver). Gebruik Ghostview om ps⇒eps uit te voeren. Importeer de eps file in je LaTeX document m.b.v. het graphics package, ik geloof dat je daar iets moet zeggen als

```
\includegraphics[width=\textwidth, clip=true]{plaatje.eps}
```

`clip` moet erbij vanwege baggerpostscript. In Excel moet je waarschijnlijk nog wat rommelen met het formaat van bijschriften om alles goed te krijgen, dit is het bekende „klooi–werk", wat je nog wel kent uit je Word tijd denk ik. . .

Adobe printerdrivers blijken vaak niet te werken, omdat Adobe er een sport van maakt om correcte postscript te genereren die bij niet–Adobe producten toch niet werkt. Ik heb deze ervaring met Illustrator en GhostScript.

Ik heb inmiddels echter mijn werkwijze veranderd en doe het nu als volgt:

Exporteer je data naar een textfile en schrijf een GNUplot scriptje ongeveer als volgt:

```
reset
set term eepic
set output 'figuur1.fig'
set key bottom
set xlabel '$I_{Basis}$'
set ylabel '$I_{Emitter}$'
plot 'data.dat' using 1:2 title 'Lijn 1' with lines,\
'data.dat' using 1:3 title 'Lijn 2' with lines,\
'data.dat' using 1:4 title 'Lijn 3' with lines
```

De gegenereerde file kun je zo importeren met

```
\resizebox{\textwidth}{!}{\input{figuur1.fig}}
```

of gewoon `\input{figuur1.fig}`, vergeet niet `\usepackage{eepic}` in je preamble te zetten. Mooi vind ik dat nu het normale font ook in je grafieken wordt gebruikt.

Als laatste kan ik je toch aanraden om „gewoon" LaTeX te schrijven, met de documentatie van

`http://ee-staff.ethz.ch/~oetiker/lshort/`

lukte het mij in een avond. GNUplot documentatie is te vinden op

`http://www.cs.dartmouth.edu/gnuplot_info.html`

vraag ***The return of*** **plaatjes**

Ik stelde niet zo lang geleden een vraag over de kwaliteit van plaatjes. Ik heb er inmiddels een oplossing voor, omdat me ook opviel dat de kwaliteit van de letter niet helemaal top was. Ik vermoet dat mijn instellingen stonden op (de default?) 150dpi, met texconfig omgezet naar 600dpi. Nu prima plaatjes en tekst.

Ik gebruik LyX onder Linux (en probeer gaandeweg meteen ook „echt" LaTeX te leren ;-). Ik vraag met het volgende af. Wat is precies het doel/nut van floats? In LyX kan in kiezen tussen een Figure float en een Wide figure float, daarin plak je dan vervolgens het echte plaatje. Is het ook mogelijk om twee plaatjes naast elkaar op te nemen (elk met een eigen onderschrift)?

P.S. is er een goed LaTeX leerboek (voorzien van een hoop voorbeelden). Heeft iemand eventueel titel en ISBN?

antwoord Door Siep Kroonenberg `<Siep.Kroonenberg@WKAP.NL>`:
Zet beide plaatjes in een minipage:

```
\begin{minipage}{100mm}
  \begin{figure}[H]
  \includegraphics...
  \caption{...}
  \end{figure}
\end{minipage}\hspace{10mm}\begin{minipage}{100mm}
  \begin{figure}[H]
```

```
  \includegraphics...
  \caption{...}
  \end{figure}
\end{minipage}
```

Door Edwin Drost <edro@OCE.NL>: **antwoord**
Floats zijn objecten (bijvoorbeeld figuren of tabellen) die je graag in de buurt van een stuk tekst wilt hebben. Toch wil je deze floats vaak niet midden in dit stuk tekst hebben. LaTeX probeert het zo te regelen dat aan beide criteria wordt voldaan. Let maar eens op, floats worden vaak bovenaan de pagina geplaatst waar je ze invoegt en als je veel floats tegelijk gebruikt maakt LaTeX een pagina met alleen maar floats.

Net zoals in een „echt" boek dus...

Voor wat betreft een goed boek over LaTeX heb ik goed nieuws, voor beginners is dat gratis en het heet *The not so short introduction to LaTeX2e* en is te downloaden van http://ee-staff.ethz.ch/~oetiker/lshort/

Met dit als leidraad en de nodige hulp van deze mailinglijst heb ik mijn afstudeerverslag geschreven.

P.S. Ik heb trouwens laatst ook eens zitten spelen met LyX en het viel me op dat de gegenereerde code niet echt op pure LaTeX leek, ik denk dat je een flinke pens met werk hebt om zo'n document nog om te zetten. LyX heeft geloof ik zijn eigen \documentclass–en met allerlei eigen commando's die natuurlijk niet in *The not so short...* gedocumenteerd zijn.

Page X of TotalPages **vraag**

Hoe kan ik de paginanummering zodanig krijgen dat Page X of TotalPages als paginanummer wordt gegenereerd. Dus bijvoorbeeld „page 14 of 25"

Ik dacht al aan fancyheaders, maar daar ben ik niet zo in thuis.

Door Piet van Oostrum <piet@cs.uu.nl>: **antwoord**
Daar staat het ook in genoemd:

```
\usepackage{fancyhdr,lastpage}
\pagestyle{fancy}
\cfoot{Page \thepage\ of \pageref{LastPage}}
```

Caption uitlijnen **vraag**

Ik gebruik in LyX een table–float en heb ingesteld dat de table niet gecentreerd wordt, maar links uitgelijnd. Ik krijg het alleen niet voor elkaar om de caption ook links uit te laten lijnen. Iemand een suggestie?

Door Piet van Oostrum <piet@CS.UU.NL>: **antwoord**
Met een truc, waarbij je wel een overfull hbox krijgt:

```
\caption{dit is een figuur\hspace*{\linewidth}}
```

Listings, babel, dutch en today **vraag**

Ik gebruik de twee pakketten listings en babel met als optie dutch. Nu wil het dat mijn header informatie waarin \today wordt gebruikt wordt verminkt op de pagina's waar listings gebruikt wordt. (extra character(s) tussen 20 en mei).

Door Piet van Oostrum <piet@CS.UU.NL>: **antwoord**
Dat heeft een bekende klank. listings rotzooit nogal met catcodes en zo, en kennelijk staat er dan bij een pagina–overgang iets niet goed. Eigenlijk zou in de headers alles weer standaard gezet moeten worden maar kennelijk wordt er iets over het hoofd gezien.

Ik heb het even nagetraced en het probleem blijkt hem in ~ te zitten. `listings.sty` herdefinieert ~ en die wordt in `\today` gebruikt.

Je kunt er omheen met:

```
\edef\vandaag{\today}
```

en dan `\vandaag` gebruiken in plaats van `\today`.

Dit moet wel na je `\begin{document}`

vraag **Doorstrepen**

Ik ben bezig met een aantal sheets en heb een probleem. Ik heb een aantal vergelijkingen, waarin delen van de vergelijking tegen elkaar weggestreept kunnen worden. Ik wil door deze delen een streep halen. Ik ken alleen geen functie binnen TEX die dit automatisch voor doet, kan iemand mij helpen.

antwoord Door Hans Hagen <pragma@WXS.NL>:

We zullen maar zeggen „het werkt”:

```
\def\hack#1%
  {\bgroup
   \setbox0=\hbox{#1}%
   \setbox2=\hbox to \wd0{\hrulefill}%
   \hbox to \wd0{\box0\hskip-\wd2\raise.5ex\box2}%
   \egroup}
```

`\hack{oeps}` ⇒ ~~oeps~~

Binnen CONTEXT is er overigens een commando `\doorstreep`, dat het iets netter doet en bovendien nest, zodat ~~oeps oeps oeps~~ de middelste dubbel doorstreept, en zo nog wat geintjes.

dit laatste werkt dus niet! Ha! ~~Hans Hans Hans~~

vraag **Compileren met** CONTEXT

Ik krijg bij het compileren het volgende:

```
TeX capacity exceeded, sorry [pool-size 97367]
```

antwoord Door Hans Hagen <pragma@WXS.NL>:

Hier gebruik ik in `texmf.cnf`:

```
main_memory       = 1500000  % 1500000  % 2500000
extra_mem_top     = 1000000  % 1000000
extra_mem_bot     = 1000000  % 1000000
font_mem_size     =  250000  %  500000
font_max          =     750  %     750
hash_extra        =   25000  %   50000

pool_size         =  400000  %  500000
string_vacancies  =   25000  %   25000
max_strings       =   50000  %   50000
pool_free         =  375000  %  475000

trie_size         =   64000
hyph_size         =    1000
buf_size          =   10000
nest_size         =     750
max_in_open       =      15
```

```
param_size        =    5000
save_size         =   20000
stack_size        =    5000
```

Zowel de tetex, fptex, 4tex en miktex distributies zijn ConTeXt aware en regelen de instellingen zelf.

Taal in ConTeXt format **vraag**

Hoe stel ik language in, en wat is een goede bufferwaarde?

Door Hans Hagen <pragma@WXS.NL>: **antwoord**

Als je `cont-en` *cq* `cont-nl` *cq* `cont-de` *cq* `cont-uk` verwerkt, is de de betreffende interface taal tegelijk default typesetting taal.

Je compileert nooit `context.tex`, maar altijd `cont-**` waarbij ** staat voor de twee letters van de taal.

Als je `texexec` draaiend hebt, kun je `texexec --make en nl` aanroepen en dan wordt het format genereerd. In geval van tetex en fptex zijn er ook de installatie scripts.

Hoe converteer je van TeX naar PDF naar Word? **vraag**

Door Niels Moes : **antwoord**

Over het Word–probleem hebben wij het vaak gehad. Onze oplossing is: begin er niet aan. Temeer omdat alle uitgevers, congresbureaus, et cetera het pdf–formaat accepteren. Afgelopen jaar heb ik telkens na enig drammen en dreinen (hetgeen je niet moet onderschatten) bereikt gekregen dat ik pdf of ps mocht indienen.

Door Siep Kroonenberg <Siep.Kroonenberg@wkap.nl>: **antwoord**

Dat klopt misschien als het alleen maar uitgedraaid moet worden, maar als de uitgever het opnieuw wil opmaken volgens de eigen standaard dan is pdf nauwelijks een optie. Op `www.tug.org/interest.html` staan enkele links voor export naar html, wat misschien een optie is.

Je kunt Postscript omzetten naar pdf met liefst Distiller maar eventueel ook Ghostscript (`-sDEVICE=pdfwrite`). Of er voor `eepic` een meer rechtstreekse route is weet ik niet. In geval van een `eps` is het nodig om een paginadefinitie toe te voegen of te wijzigen die de pagina gelijkmaakt aan de boundingbox. Het volgende perl–scriptje van Sebastian Rahtz zou dat moeten doen:

```
#!/tool/bin/perl
# fitps: a script to slightly transform an EPS file so that:
#   a) it is guarenteed to start at the 0,0 coordinate
#   b) it sets a page size exactly corresponding to
#      the BoundingBox
# This means that when GhostScript renders it, the result
# needs no cropping. GhostScript can then write a graphic
# file direct without the tedious pnmcrop etc. It needs a
# Level 2 PS interpreter.
#
# If the bounding box is not right, of course, you have
# problems...
#
# The only thing I have not allowed for is the case of
# "%%BoundingBox: (atend)", which is more complicated.
#
# Sebastian Rahtz, for Elsevier Science
```

```
# adapted by Siep Kroonenberg for Activeware Perl
# avoiding redirection
$filedate="1996/07/10";
$fileversion="1.1";
if (!$ARGV[1]) { die "Usage: fitps infile outfile"; }
$Filename=$ARGV[0]; $Outfile = $ARGV[1];
if (!open(TMP,"<$Filename") || !open(OUTF,">$Outfile")) {
  die "Couldn't open input- or output file";
}
$bbneeded=1;
local $bbpatt="[0-9\.\-]";
while (<TMP>) {
 if ( /%%: (atend)/) {
        $bbneeded = 0;
        print OUTF;
        }
 elsif
 ( /%%BoundingBox:($bbpatt+)\s($bbpatt+)\s($bbpatt+)\s($bbpatt+)/ )
 {
# only read the *first* bounding box
     if ($bbneeded) {
         $width = $3 - $1;
         $height = $4 - $2;
         $xoffset = 0 - $1;
         $yoffset = 0 - $2;
    print OUTF "%%: 0 0 $width $height\n";
    print OUTF "<< /PageSize [$width $height] >> setpagedevice\n";
    print OUTF "gsave $xoffset $yoffset translate\n";
         $bbneeded=0;
     }
# else ignore that embedded BoundingBox anyway
 }
else
 {    print OUTF;  }
}
close(TMP);
print OUTF "grestore\n";

;
```

antwoord Door Edwin Drost <edro@OCE.NL>:
Word *is* een kuil vol slangen, ik ben niet voor niets aan LaTeX begonnen, als ik trouwens het gekanker van mijn omringende collega's hoor wil ik momenteel helemaal niet meer terug.

Ik zou het trouwens wel leuk vinden als Compaq ook bekend maakt aan welke specifieke applicaties gebruikers zich ergeren.

(zie: http://www.mori.com/polls/1999/rage.htm)

vraag **Problemen met dvips**
Het volgende probleem doet zich voor:

Bij het dvips–en van een document van ongeveer 200 pagina's met bijdrages van bijna twintig verschillende auteurs, met ieder hun eigen opvatting over hoe je een LaTeX document maakt, krijg ik de volgende waarschuwing:

```
Page 97 may be too complex to print
```

Als ik de ps file vervolgens met ghostview bekijk ontbreken in de linkerbalk de paginanummers. Als ik elk der bijdrages apart behandel ontmoet ik geen enkel probleem. Door de verschillende auteurs is het aantal aangeroepen style files nogal groot.

Heeft iemand een idee in welke richting ik het moet zoeken?

Door Hendri Hondorp <hendri@CS.UTWENTE.NL>: **antwoord**
optie `-a1` gebruiken in `dvips`.

Inmiddels gechecked bij de vragensteller en deze werkt.

Caption tekst **vraag**
Hoe kan ik een hele caption (Figuur. . .+ tekst) zetten in `\small \textit`.

Door Maarten Gelderman <mgelderman@BIK.ECON.VU.NL>: **antwoord**
Even los van de andere antwoorden, kijk eens naar het package `caption`. Met dingen als

```
\usepackage[small,bf]{caption}
```

krijg je automatisch kleine bijschriften waarin tabel nr vet staat en het lettertype van het geheel klein is. Er zijn nog veel meer opties, maar dit is mijn standaard, dus de rest ken ik niet uit mijn hoofd.

Grote vragen, kleine antwoorden **vraag**
Het zal wel erg tegen de regels van de typografie zijn maar toch: In mijn wiskunde opgaven bundels wil ik de antwoorden aanzienlijk kleiner zetten dan de rest van de tekst. Een ander font(grootte) kiezen dat lukt me wel, maar de regelafstand verkleinen en vooral de wiskunde teksten kleiner krijgen hoe moet dat?

Ik gebruik NTG–class `rapport3` in 12pt, voor de wiskunde teksten gebruik ik het euler font en om de antwoorden achterin het boekje te krijgen gebruik ik `exercise.sty`.

Door Wybo Dekker <wybo@SERVALYS.HOBBY.NL>: **antwoord**
Je kleine tekst in een minipage of een parbox zetten:

```
\documentclass[12pt]{article}
\begin{document}
\\[2ex]
\begin{minipage}{100mm}
\scriptsize\baselineskip-6pt
dit is een gewone tekst waartussen we zo een
blokje klein gaan zetten, zodanig dat ook de
regelafstand kleiner wordt dit is een gewone
tekst waartussen we zo een blokje klein gaan
zetten en wel zo dat de regelafstand kleiner
wordt dit is een gewone tekst, waartussen we
zo een blokje klein gaan zetten, zodanig dat
als het lukt de regelafstand kleiner wordt!!
\end{minipage}
\end{document}
```

Door Piet van Oostrum <piet@CS.UU.NL>: **antwoord**
Maar dan wordt er geen pagina–afbreking gedaan. De clou is dat je een `\par` moet zetten aan het eind van het stuk dat kleiner moet. TEX gebruikt namelijk de instellingen die bij het eind van de alinea gelden.

```
{\small
Antwoorden
\par
}
```

of:

```
\begin{small}
Antwoorden
\par
\end{small}
```

antwoord Door Hans Hagen <pragma@WXS.NL>:
Een ConTeXt oplossing:
Eerst wat blokken definiëren (die we niet verbergen):

```
\definieerblok[antwoord]
```

Die moeten klein:

```
\stelblokin[antwoord][binnen={\switchnaarkorps[klein]}]
```

Dan een nummering:

```
\doornummeren[antwoord]
```

Nu de tekst

```
\starttekst
```

weet ik wat voor'n tekst

```
\beginvanantwoord
\antwoord een antwoord \par
\eindvanantwoord
```

en nog wat tekst
En nu nog aan het eind nog een keer de antwoorden

```
\stelblokin[antwoord][binnen]


\gebruikblokken[antwoord]
```

Ik zou vragen ook in een blok zetten

vraag **A2 PostScript vraag**
Een collega van mij (dezelfde van mijn vorige PostScript vraag) zit (nog steeds) met zijn A2 PostScript file. De verkleining met psresize werkt, maar nu wil hij de A2 file in stukken verdelen over A4–tjes om een idee te krijgen van het uiteindelijke resultaat. Iemand met ideeën?

antwoord Door Taco Hoekwater <taco.hoekwater@WKAP.NL>:
Hm, iets voor Siep lijkt me maar die is helaas met vakantie. pnmposter lijkt me de „best bet".

http://fsinfo.cs.uni-sb.de/~stahl/pnmposter/index.html

antwoord Door Piet van Oostrum <piet@CS.UU.NL>:
Het programma poster. Op CTAN in support/poster.

Wedden dat ik durf? **vraag**

In een tekst (in een times–variant) krijg ik een ongewenste botsing, en soort ligatuur, in de zin ,,wedden dat ik durf?" In ,,mijn" Times reikt de f bovenin naar rechts, het vraagteken reikt juist boven naar links en daar overlappen ze elkaar. Ik ga dat ,,oplossen" met een `{}` tussen de f en het vraagteken.

Is dat een *goeie* oplossing of zou dat anders/beter moeten?

Door Taco Hoekwater `<taco.hoekwater@WKAP.NL>`: **antwoord**

Je kunt het wel doen, maar het helpt geen zier :-). Het is namelijk geen ligatuur, maar een uit–stekend karakter. Wat wel werkt of zou moeten werken is `durf\/?` (niet alleen schuine fonts hebben een italic correction).

Door Frans Goddijn `<frans@iaf.nl>`: **antwoord**

Dankje! Ik heb deze oplossing ook op een andere plaats kunnen gebruiken: in een hoofdstuktitel stond (in vet, schreefloos) de naam ,,Offermans" en daar viel op scherm en in print de ff–ligatuur gewoon weg!

`Of{}fermans` hielp niet, `Of\/fermans` wel!

Door Maarten Gelderman `<mgelderman@BIK.ECON.VU.NL>`: **antwoord**

`Of\kern 0pt fermans` lijkt me nog iets mooier. Het probleem met je accolades is dat die verdwijnen als het geheel door het afbreekmechanisme van TEX gaat, met een kern gebeurt dat niet.

Door Sven Bovin `<sven@HARTREE.CHEM.KULEUVEN.AC.BE>`: **antwoord**

Ik denk dat om een ligatuur te vermijden er eerder een (lege) `\mbox{}` moet staan daar — als ik mij niet vergis — het invoegen van ligaturen gewone `{}` gewoon wegstript.

Door ter Haar Romeny `<romeny@letmail.let.leidenuniv.nl>`: **antwoord**

Het bijzondere van TEX is dat het losse invoer als `ff` en `fl` kan herkennen als tekst die in de klassieke typografie met een enkel stukje lood wordt gezet, een combinatie van de twee (soms zelfs drie: `ffl`) letters. ff, fl en ffl. TEX maakt van dat soort reeksen dan ook een enkel karakter (indien het font daarover beschikt en een goede ligaturentabel is gemaakt). Als je in deze gevallen wilt voorkomen dat TeX van `ff` een enkel karakter maakt, is een leeg commando of eventueel de bij roman fonts toch lege italic correction (`\/`) een oplossing.

De reeks f + vraagteken is echter niet zo'n klassieke combinatie. De genoemde oplossingen zullen je dus ook niet helpen. De overlap is hier het resultaat van niet helemaal perfecte digitalisering van het font: na de f hoort het font meer ruimte te geven als bepaalde karakters volgen, en kennelijk gebeurt dat niet. Je zult een klein beetje extra wit moeten toevoegen met bijvoorbeeld `$\,$`.

KoortsTEXmometer **vraag**

Ik wil een graden (temperatuur) tekentje invoegen. Is er standaard LATEX commando dat ervoor zorgt dat het wat hoger komt te staan en dergelijke. Is er een overzicht van dat soort bijzondere tekens?

Door L.Hoekema@BIOCH.UNIMAAS.NL : **antwoord**

Ik heb het volgende commande gemaakt:

```
\newcommand{\degrees}{\ensuremath{\:^{\circ}}C\xspace}
```

Dus als je typt `37\degrees` krijg je automatisch een klein beetje ruimte tussen 37 en het graden teken, bovendien blijven 37 en `\degrees` altijd op 1 regel bijelkaar.

antwoord Door Piet van Oostrum <piet@CS.UU.NL>:

```
\usepackage{textcomp}
...
\textdegree of \textcentigrade % (geeft ook een C)
```

of:

```
$^\circ$
```

of:

```
\usepackage{SIunits}
...
\degree of \degreecelcius
```

vraag **PostScript: no %%% Page comments generated?**
In sommige gevallen roept DVIPS (5.86) bij het maken van PostScript uitvoer:

```
Warning: no %%Page comments generated
```

In welke gevallen gebeurt dat, en hoe kan ik dat voorkomen? Ik wil ze wel hebben omdat anders bv. Ghostscript alleen maar vooruit kan bladeren, niet achteruit. Ook ander PS–interpreters hebben moeite met PS–code die geen pagina-markers bevat.

antwoord Door Piet van Oostrum <piet@CS.UU.NL>:
Meestal gebeurt dit als dvips denkt dat de printer te weinig geheugen heeft. Voorkomen door de geheugenoptie in `config.dvips` hoger te zetten.

vraag **Afbreken en slashes**
Ik heb wat problemen met afbreken:

Bijvoorbeeld „R2–heterozygous individuals”. Dit zou afgebroken kunnen worden tussen zy en gous. Alleen dat gebeurt gous steekt uit in de marge. Als ik via hyphenation het afbreekpatroon opgeef, dan kan ik dat alleen doen voor heterozygous (en dat helpt niet), want in `\hyphenation` mogen geen niet–letters staan.

Hetzelfde probleem heb ik met woorden met slashes erin: `protein~C/protein~S`: dit zou ergens afgebroken moeten worden, maar dat gebeurt niet. Zelfde vraag als hierboven.

antwoord Door Maarten Gelderman <mgelderman@BIK.ECON.VU.NL>:
Gewoon `\-` in het woord opnemen op de plaatsen waar het afgebroken mag worden.

En gewoon de / vervangen door het commando `\slash`

vraag **Oldstyle hack gebroken?**
Ik heb geprobeerd om de code voor geïmproviseerde old style numerals (MAPS 20, pp. 117–119) in een style bestandje te gieten en te gebruiken, eerst in een document met Computer Modern fonts, maar de bedoeling is het document uiteindelijk in Times te zetten (vandaar dus die geïmproviseerde old style numerals).

TeX komt echter klagen over een „Illegal unit of measurement” en nog een aantal zaken.

TeX verslikt zich blijkbaar niet op `\Txtnum{<...>}`, een afgeleid commando dat ik gebruik om in lijn te blijven met een aantal andere documenten.

antwoord Door Taco Hoekwater <taco.hoekwater@WKAP.NL>:
Mijn versie van `graphicx` heeft een hekel aan de `\scalebox` zoals die in je source staat (en ik krijg dan ook precies dezelfde foutmelding). Probeer eens

```
\def\verklein{\scalebox{0.9}[0.8]}
```

dat werkt voor mij.

TEX „verslikt" zich niet in `\Txtnum`, maar het *werkt* ook niet. Als je wilt dat het wel werkt, moet je

```
\newcommand{\Txtnum}[1]{\armeluisdartels{#1}}
```

vervangen door

```
\let\Txtnum\armeluisdartels
```

Weg met die nummers

vraag

Ik heb een latexdocument met vooraan een inhoudstafel en achteraan een index. Ik wil geen paginanummers (dus `\pagestyle{empty}`). Ik krijg op pagina 1 (de eerste pagina van de inhoudstafel) toch het cijfer 1 en op pagina 50 (de eerste pagina van de index) toch het cijfer 50. De andere pagina's hebben geen nummer. Kan iemand me vertellen hoe ik beide pagina's zonder nummer krijg. (`\thispagestyle{empty}` lijkt geen oplossing)

Door Piet van Oostrum <piet@CS.UU.NL>:

antwoord

Dat laatste komt omdat ze beide zelf een `\thispagestyle{plain}` doen, en die komt na jouw `\thispagestyle{empty}`. En als je de `\thispagestyle{plain}` na de `\tableofcontents` en dergelijke zet, dan geldt dat voor de laatste pagina, niet voor de eerste.

Simpele oplossing omdat het hele document met `\pagestyle{empty}` moet:

```
\usepackage{nopageno}
```

of:

```
\makeatletter
\let\ps@plain\ps@empty
\makeatother
```

Snelle rugnummers

vraag

Pffft... zo'n middag dat de uren verstrijken zonder dat er *werk* gedaan kan worden. Ik zou „even" rugnummers printen voor de plaatselijke rijvereniging en eerst deed ik dat in CorelDraw, lekker makkelijk, maar al snel kreeg ik door dat ik dan de hele middag zou zitten p**len, even tijdrovend als makkelijk dus.

Maar ook in LaTeX ging dat niet even–snel. Het verschil: in Draw kun je meteen het eerste rugnummer printen maar moet je ook elk nummer apart gaan maken, in TEX moet je eerst nadenken maar als je het raadsel heb opgelost is de rest in een wip gedaan.

Eerste probleem was het grote font. Als ik deed:

```
\newfont{\nrkop}{tnr8 scaled 50000}
```

(in mijn setup is Times toevallig `tnr8`, dat terzijde) dan kreeg ik de melding dat dat te groot was en dat het zou worden verkleind naar 1000. Ik kreeg niet in de gaten waar nou de grens zit maar het ging meteen veel beter met

```
\newfont{\nrkop}{tnr8 at 450pt}
```

Uiteindelijk ging het heel snel en gemakkelijk zoals hieronder. Ter verstrooiing... een simpele hacker at work. Een slimmere user zou natuurlijk TEX de nummers zelf laten ophogen en plaatsen...

```
\documentclass{artikel1}
\pagestyle{empty}
\usepackage{fg,fghhtime} % eigenlijk alleen nodig voor mijn times-fontje
\oddsidemargin 0mm       % pagina groot maken op A4
\evensidemargin -10mm
```

```
\marginparwidth 5mm
\topmargin -15mm
\textheight 250mm
\textwidth 180mm
\headheight 4mm
\headsep 4mm
\def\baselinestretch{0.95} % hoeft hier eigenlijk niet
\newfont{\nrkop}{tnr8 at 450pt}  % font voor bijna alle 2-cijferige nummers
\newfont{\nrkkop}{tnr8 at 400pt} % alleen voor rugnummer 100, de laatste
\begin{document}
\newcommand{\rugnummers}[2]{#1\vfill#2\newpage}
%nummer, en daaronder weer nummer, volgende blad
\nrkop          % groot font

\rugnummers{25}{26}
\rugnummers{27}{28}
\rugnummers{29}{30}
\rugnummers{50}{51}
\rugnummers{52}{53}
\rugnummers{54}{55}
[...]
\nrkkop
\rugnummers{100}{}
\end{document}
```

antwoord Door Maarten Gelderman <mgelderman@BIK.ECON.VU.NL>:
Hoi, dat kan korter. Even afgezien van de layout, die me niet interesseert:

```
\documentclass{article}
\usepackage{multido}
\usepackage[margin=2cm,a4paper]{geometry}

\font\groot cmr10 at 450pt
\def\nummer#1{\groot #1\clearpage}
\begin{document}
\multido{\iNR=1+1}{100}{\nummer{\iNR}}
\end{document}
```

antwoord Door Hans Hagen <pragma@WXS.NL>:
Arme Frans. Je moet de MAPS lezen!
Dit gaat ook:

```
\starttekst

\stelkorpsin
  [pos]

\stellayoutin
 [rugwit=.5cm,kopwit=.5cm,
  hoofd=0pt,voet=0pt,
  breedte=midden,hoogte=midden]

\NormalizeFontWidth
  \NummerFont {100} {\tekstbreedte} {Regular}
```

```
\dorecurse{100}
 {\omlijnd
    [hoogte=14cm,breedte=ruim,offset=0pt]
    {\NummerFont\recurselevel}
  \vskip.5cm}

\stoptekst
```

Die I in 100+ is een buitenbeentje, vandaar beter:

```
{\NummerFont%
 \ifnum\recurselevel>99%
 \hskip-.25ex\fi\recurselevel}
```

Zo zie je maar, het valt best mee. Hier zouden we ook wel eens een cursusje aan kunnen wijden: truukjes in TEX.

Z.O.Z./PTO **vraag**

Dit heeft vast wel eens iemand bedacht. Op een tentamen dat uit twee pagina's bestaat maar dubbelzijdig wordt geprint zou ik rechts onderaan de eerste pagina graag de kreet Z.O.Z. vermelden. Hoe kan ik dat doen?

Door Piet van Oostrum <piet@CS.UU.NL>: **antwoord**

Ik ben net met een tentamen bezig. Ik gebruik het volgende (met een pijltje in plaats van „z.o.z." en een balkje op de laatste pagina).

```
\usepackage{fancyhdr}
\usepackage{ifthen}

\pagestyle{fancy}

\fancyhf{}
%\renewcommand{\footrulewidth}{0.4pt}
\chead{\sf \naam\ $-$ \datum}
\cfoot{\ifthenelse{\pageref{endpage}=1}%
          {\rule{12mm}{4mm}}{\thepage}}
\rfoot{\ifthenelse{\pageref{endpage}=1}%
          {}{\ifthenelse{\pageref{endpage}=\value{page}}%
          {\rule{12mm}{4mm}}{\LARGE$\Longrightarrow$}}}
\fancypagestyle{first}{\chead{}}\thispagestyle{first}
```

en aan het eind:

```
\label{endpage}
```

Door Jan Vroonhof <vroonhof@MATH.ETHZ.CH>: **antwoord**

Wij gebruiken een eigen brouwsel voor onze opgaven en tentamens. Werkt ook voor meer dan 1 blad. Het duits moet je zelf even veranderen. Helaas kent Babel `\pleaseturnstring` nog niet.

(Zet dit in je `tentamen.cls`)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%         pagestyles                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\def\ps@footings{
        \let\@mkboth\@gobbletwo
```

```
        \def\@oddhead   {}
        \def\@oddfoot   {\footline}
        \def\@evenhead  {}
        \def\@evenfoot  {\evenfootline}
        }

%                       defaults
\newcommand{\pleaseturnstring}{Bitte wenden!}
\newcommand{\seenextpagestring}{Siehe n\"achstes Blatt!}
\newcommand{\footline}{\hfil%
 {\bfseries\expandafter\@firstoftwo\botmark{}{}}}
\newcommand{\evenfootline}{\hfil%
 {\bfseries\expandafter\@secondoftwo\botmark{}{}}}
\AtBeginDocument{\markboth{\pleaseturnstring}{\seenextpagestring}}
\AtEndDocument{\markboth{}{}}
\@twosidetrue \raggedbottom

\pagestyle{footings}
```

vraag **Kolommen even breed in tabellen**

Ik wil graag een tabel met drie kolommen precies even breed hebben als de tekstbreedte. Wie weet hoe ik een deling kan uitvoeren, zodat ik iets krijg als.

```
\begin{tabular}{| p{\textwidth / 2} |
                 p{\textwidth / 4} |
                 p{\textwidth / 4} |}
    ...
\end{tabular}
```

antwoord Door Adri van der Meer <A.W.J.vanderMeer@MATH.UTWENTE.NL>:
Dat gaat niet zonder meer goed: je moet ook nog rekening houden met het feit dat er ruimte tussen de kolommen zit. Ik heb het wel eens zo gedaan:

```
\newlength{\ceen} \setlength{\ceen}{0.5\textwidth}
\addtolength{\ceen}{-2\tabcolsep}
\newlength{\ctwee} \setlength{\ctwee}{0.25\textwidth}
\addtolength{\ctwee}{-2\tabcolsep}
\newlength{\cdrie} \setlength{\cdrie}{0.25\textwidth}
\addtolength{\cdrie}{-2\tabcolsep}
```

. . . en dan kun je `\ceen` enzovoorts gebruiken als kolombreedtes.

antwoord Door Hans Hagen <pragma@WXS.NL>:
In ConTeXt:

```
\starttabulatie[|p|p|p]
\NC whatever text \NC whatever text \NC whatever text \NC\NR
\stoptabulatie
```

also breaks across pages.

vraag **Fontwijziging in tabular**

Is er een simpele manier om in een regel van een tabular een fontwijziging te maken? Met name in de eerste regel.

Het gaat me om het foolproof maken van de situatie waarin de kopregel van de tabel altijd `\bfseries\small` moet zijn, en de rest van de tabel niet. Ik wil een soort van macro `\kopregel` maken die dat doet, ongeacht het aantal kolommen.

Door Piet van Oostrum <piet@CS.UU.NL>: **antwoord**

```
\newcommand{\inskoptxt}{\bfseries\small}
\newcommand{\kopregel}[1]{\koprrr#1&\koprrr}
\def\koprrr#1&#2\koprrr{\inskoptxt #1\def\testrrr{#2}\ifx\testrrr\empty
  \def\nextrrr{\\}\else\def\nextrrr{&\koprrr#2\koprrr}\fi\nextrrr}
```

Hans Hagen
PRAGMA ADE
Ridderstraat 27
8061GH Hasselt NL
pragma@wxs.nl

sgml

The NTG MAPS bibliography
from SGML to TEX to PDF

abstract
A few years ago, the ntg decided to put their Maps volumes on the internet in the pdf file format. At about the same time, it was decided to build the associated bibliography in such a way that it could be used to produce both a html and pdf document.
Recently, the Maps bibliography has been converted to a proper xml document source. In the process, the descriptions were made as consistent as possible. The xml source was used as input for a pdf document, that provides extensive browse and search options. This pdf file, along with the Maps articles, is provided to ntg members as an additional service.
In this article the electronic ntg Maps will be presented and the specific characteristics of the production process will be explained. Also, some of the complicating aspects will be discussed. I assume that the reader is familiar with sgml and TEX. The focus will be on the interface between sgml, TEX and pdf.

keywords
cdrom, ConTEXt, ntg, Maps, pdf, sgml, xml

Introduction

The Dutch speaking TEX Users Group NTG was founded over ten years ago. Right from the start, the minutes and appendices (MAPS) of the meetings have played an important role not only in registering what was taking place, but also in providing useful information to the members on how to use TEX and where to find macros and programs.

During those ten years the MAPS, which is sent to the members two times a year, has grown to about 200 pages per volume. At the tenth anniversary it was decided to redesign the layout and change the name into a meaningless succession of four characters. From that moment on, the articles became the MAPS and the minutes got their own A5 booklet. One reason for splitting up the content was that pure membership issues were not that relevant for non members who could fetch the latest volumes from CDROM's and the NTG internet site.

When the MAPS was made available on the NTG internet site, a bibliography was set up by Erik Frambach. This first version was coded in the BIBTEX format and converted to HTML afterwards. Erik also provided a version coded in TEX, that was used to construct a typeset PDF version.

When maintenance of the bibliography was transferred to the MAPS editors, Taco Hoekwater converted the database into XML. It was this database that provided the input for the PDF document I will describe here in more detail. This document does not differ that much from the prototype version. In the process of fine–tuning the document, I also normalized the index entries and author names.

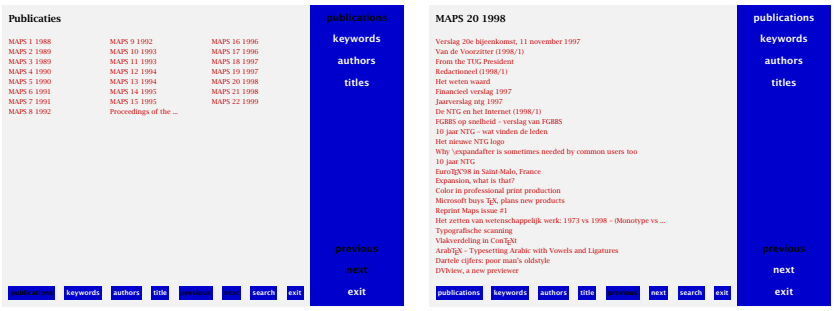
The structure

When we open the bibliography, the title page (as shown in figure 1) shows up. The abstract is typeset in black, but other components come in red, white, and blue: the colors of the dutch national flag. The belgian flag is in black, yellow, and red, and the yellow shows up when we click on a button.

title page article description

Figure 1

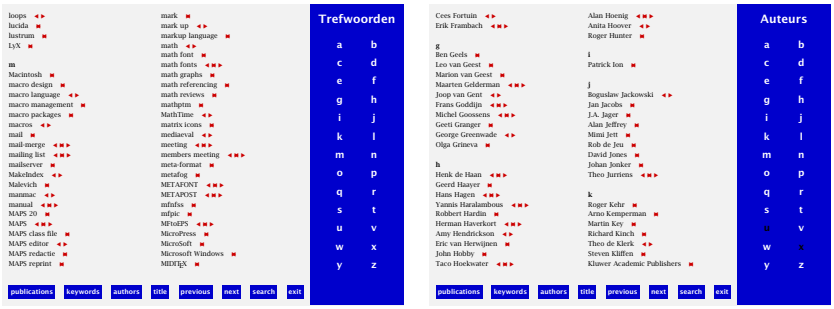
The right side and bottom areas of the screen are used for navigational ornaments. In a document like this one will, in most cases, use indices to locate a topic. Nevertheless, we provide some tables of contents: a main one and one per volume (see figure 2).



main table of contents volume table of contents

Figure 2

There are three indices, of which the keyword and author index are linked registers (figure 3). A linked index permits the user to 'walk though the document', i.e., showing each page to which the index entry refers.



keyword index author index

Figure 3

The title index (figure 4) collects the articles per author. Finally, there is a colofon page that tells some more about the NTG. Like the MAPS, the bibliography is a mix of dutch and english.

title index colofon

Figure 4

The Database

The database is coded in XML, a restricted form of SGML that was introduced recently as something new and better, and seems to be accepted more easily. The 22 volumes that make up the document described here have 710 abstracts. The resulting document has 786 pages and 18,421 hyperlinks. The PDFTEX version 14a binary packs this into a 4.5 Megabyte PDF document. Although not that complicated and big, this document is a nice example of what TEX can do with such a database.

The abstract shown in figure 1 is coded in XML as follows:

```
<bibentry type="article" id="20-44" language="en">
  <title>
    The Calculator Demo -- Integrating &tex;, MetaPost, JavaScript and PDF
  </title>
  <author>
    <au index="hagenh"><fn>Hans</fn><sn>Hagen</sn></au>
  </author>
  <published volume="20" year="1998" pages="290-296" size='201'>
    <journal>MAPS</journal>
  </published>
  <keywords>
    <key>METAPOST</key> <key>JavaScript</key> <key>PDF</key>
    <key>&context;</key> <key>pdf&tex;</key>
  </keywords>
  <abstract>
    Due to it's open character, &tex; can act as an authoring tool. This
    article demonstrates that by integrating &tex;, MetaPost, JavaScript
    and PDF, one can build pretty advanced documents. More and more
    documents will get the characteristics of programs, and &tex; will be
    our main tool for producing them. The example described here can be
    produced with pdftex as well as traditional &tex;.
  </abstract>
</bibentry>
```

Some logo's are coded as special tokens using the `&token;` syntax. Certain entries come with key–value pairs. Although work is in progress by Taco Hoekwater to let a special version of TEX directly process such input, for the moment, we convert it into something more natural to TEX.

```
\endSGML[bibentry]
\begSGML[bibentry][type=article,id=20-44,language=en]
\begSGML[title]The Calculator Demo -- Integrating \tokSGML{tex}, MetaPost,
JavaScript and PDF\endSGML[title]
\begSGML[author]
\begSGML[au][index=hagenh]
\begSGML[fn]Hans\endSGML[fn]
```

```
\begSGML[sn]Hagen\endSGML[sn]
\endSGML[au]
\endSGML[author]
\begSGML[published][volume=20,year=1998,pages=290-296,size=201]
....
```

The conversion is straightforward and takes only a few lines of PERL. All manipulations will be done in CONTEXT. Before we will uncover some of the details of this manipulation, I explain the way the layout and basic struture is defined.

Layout and basic structure

A screen has an aspect ratio quite different from the average paper dimensions, like A4 and letter. Here we use one of the predefined paper sizes S6, that has a width of 600pt and an aspect ratio of 4:3. We use the same size for typesetting and printing, which means that the page is automatically cropped to the paper size we print on.

```
\setuppapersize [S6] [S6]
```

We don't use headers, footers, and top areas, but only the text and bottom ones. The bottom as well as right edge are used for navigational tools.

```
\setuplayout
  [topspace=10pt, backspace=10pt,
   width=440pt, height=fit, header=0pt, footer=0pt,
   rightedge=120pt, rightedgedistance=20pt, margin=0pt,
   bottomdistance=10pt, bottom=20pt]
```

The bibliography is typeset in 10 point Lucida Bright, a font that renders very well on screens. We preload symbolset `navigation 1` that is a subset of CONTEXT navigation symbol font. This font is part of the standard CONTEXT distribution.

```
\setupbodyfont  [lbr,10pt]
\usesymbols     [nav]
\setupsymbolset [navigation 1]
```

We turn on colors. To save some processing time we use local color mode, which means that bookkeeping is minimized to page bound coloring.

```
\setupcolors [state=local]
```

The page has a light gray (.95) screen as background. The rightedge text and bottom areas become blue. The offset ensures that the text will not touch the border.

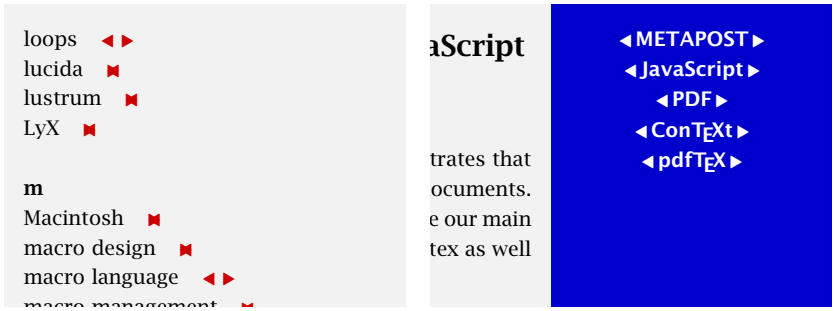
```
\setupbackgrounds
  [page]
  [background=screen]

\setupbackgrounds
  [text,bottom] [rightedge]
  [background=color, backgroundcolor=darkblue, backgroundoffset=10pt]
```

Because we are dealing with an interactive document, we have to set up the interaction. In a document like this we can save quite some bytes when we use page destinations in hyperlinks instead of named ones. By saying `page=yes` we tell CONTEXT to use page destinations when possible. Because we will use a few menus, we enable this feature. Especially when we use color, a (bold) sans serif font makes hyperlinks more visible. Hyperlinks will be colored red, unless they point to the current page, in which case they will be typeset in black.

```
\setupinteraction
  [state=start,
   page=yes,
   menu=on,
   style=\ssbf,
```

```
  symbolset=navigation 1,
  color=darkred,
  contrastcolor=black]
```



linked keyword keyword menu

Figure 5

The indices are, as already mentioned, cross linked ones. This is a standard ConTeXt feature that was first applied in 1996 in a huge 4000 page document with many indices and lists (different tables of contents). Such linked indices are a useful alternative for endless lists of page numbers in an index. The registers are defined by:

```
\defineregister [keyword]        [keywords]
\defineregister [author]         [authors]
\defineregister [titlebyauthor] [titlesbyauthor]

\setupregister  [keyword,author] [coupling=yes]
```

We must set up the cross linked registers explictly. The next commands load and preprocess the relevant data.

```
\coupleregister [keyword]
\coupleregister [author]
```

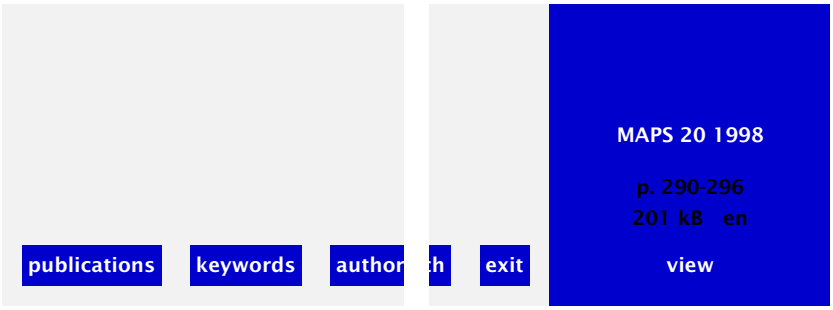
The titles by author index is not cross linked. Here we want to click on the whole entry, and we don't want to see a page number or symbol (figure 8). There is no alphabetic section indicator (a, b, c, etc.). We will expand index entries (more about that later) and limit their typeset size.

```
\setupregister
  [titlebyauthor]
  [n=1, symbol=none, interaction=text, indicator=no]

\setupregister
  [keyword,author,titlebyauthor]
  [maxwidth=.8\hsize, expansion=yes]
```

The keywords themselves are accompanied by left and right triangles that bring us to the previous or next location. Clicking on the keyword itself brings us back to the index. In the index there can be one, two, or three symbols that represent the begin, middle, and/or end of the list.

In ConTeXt one can define menus that can be turned on and off. Normally the menus are located in the top, bottom, left, and/or right edge areas. There can be multiple stacked or overlapping menus. Here we use one bottom menu and three right menus. When available, we will put some additional information on the article in the bottom of the menu (figure 6).



bottom menu status area

Figure 6

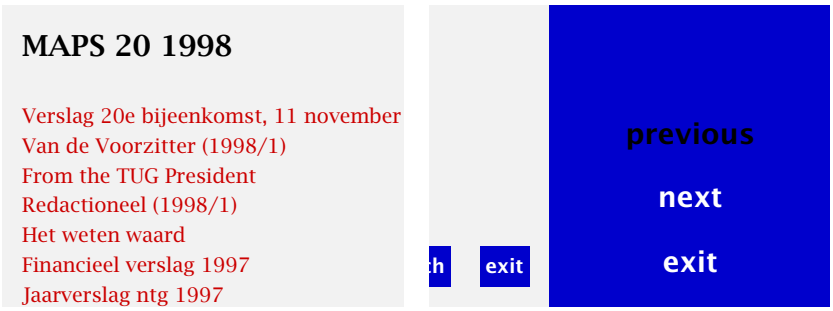
We use no fancy buttons, only colored backgrounds, so we can stick to:

```
\setupinteractionmenu
  [bottom]
  [state=start,
   height=\bottomheight,
   frame=off,
   background=color,
   backgroundcolor=darkblue,
   color=white,
   style=\ssbf]
```

These settings will be applied to:

```
\startinteractionmenu[bottom]
  \but [publications]     publications \\
  \but [keywords]         keywords     \\
  \but [authors]          authors      \\
  \but [titles]           titles       \\
  \but [previoussubpage] previous     \\
  \but [nextsubpage]      next         \\
  \but [SearchDocument]   search       \\
  \but [CloseDocument]    exit         \\
\stopinteractionmenu
```

ConTeXt has a so called integrated cross reference mechanism. Instead of providing dozens of commands that deal with interactive issues, we use only a few that interpret their arguments. In this example, the last two references are special in the sense that they refer to viewer actions (figure 7).



simple lists viewer control

Figure 7

At the right of the screen there are a few alternative menus. These are laid over each other and turned on and off when needed. The index menu (the one with the alphabet)

contains a vertical box with a..z buttons, each taking 40% of the width, and typeset as a centered paragraph. (figure 8). The main menu is set up as:

```
\setupinteractionmenu
  [right]
  [state=start,
   frame=off,
   color=white,
   style=\ssbfb,
   distance=overlay]
```

By setting `distance` to `overlay`, the right menu will be typeset on top of other menus we place at the right (otherwise menus would be stacked). The definition below shows that occasionally one cannot do without building boxes and using fills. Even when one uses a high level of abstraction as provided by CONTEXT, some insight in `\xbox`, `\xfill` and `\xskip` (with `x` being `v` or `h`) is useful.

```
\startinteractionmenu[right]
  \boxofsize \vbox \textheight \bottomdistance \bottomheight
    \bgroup
      \setupalign[middle]
      \but [publications]    publications \\
      \but [keywords]        keywords     \\
      \but [authors]         authors      \\
      \but [titles]          titles       \\
      \vfill
      \but [previoussubpage] previous     \\
      \but [nextsubpage]     next         \\
      \but [CloseDocument]   exit         \\
      \unskip
    \egroup
\stopinteractionmenu
```

The `\unskip` is needed because by default some white space is added between buttons and by using an embedded `\vbox`. CONTEXT cannot automatically correct for this at the bottom of the menu.

The second type of structural element (apart from the registers) is the table of contents. Like registers, there can be many of them at each level of sectioning. For readability, we don't use chapters and sections but:

```
\definehead [structuralelement] [chapter]
\definehead [publication]       [chapter]
\definehead [publicationtitle]  [section]
```

They will show up as defined below. We just set up their parents from which they inherit. Numbers don't make sense here.

```
\setuphead
  [chapter,section]
  [style=\bfb,
   before=,
   after={\blank[3*medium]},
   align={right,broad},
   number=no]
```

In CONTEXT there currently are three levels of page numbering: the real page number which runs from 1 to the number of pages in the document, the typeset number that in most cases does not increment on unnumbered pages, and the subpage number, that is used in navigational tools. Some day this scheme will be extended to a more advanced one with more (independent) subpage numbers. Here we use the subpage numbers on a per chapter base:

```
\setupsubpagenumber [state=start, way=bychapter]
```

When we define (or actually clone) a head, we also get a dedicated list similar to a table of contents. In this document, we will typeset this list with the following settings. We use a simple title–only list (figure 7).

```
\setuplist
  [publication,publicationtitle]
  [alternative=f,
   interaction=all,
   maxwidth=.8\textwidth,
   before=,after=]
```

By default heads are not expanded, but when we use variables, which happens to be the case in this kind of SGML processing, we definitely need to expand this variable when we write it to a list.

```
\setuphead [publication,publicationtitle] [expansion=yes]
```

A button is something to click on. Menu buttons are typeset conforming to the menu settings, and those not native to menus show up like:

```
\setupbuttons [color=white, style=\ssbfb, frame=off]
```

The last layout item we will setup before dealing with SGML are the menus that show up along side a bibliography item and the registers.

```
\defineinteractionmenu
  [publication][right]
  [color=white,
   contrastcolor=white,
   frame=off,
   style=\ssbf,
   distance=overlay]
```

The menus alongside the registers will show a navigational alphabet. The next definition makes sure we get big buttons with a width of 40% edge width. For consistency reasons, the struts follow the current font outside the button, so here we have to make sure we use a larger button.

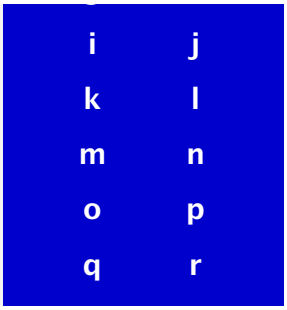
```
\defineinteractionmenu
  [navigation][right]
  [color=white,
   contrastcolor=white,
   distance=overlay,
   width=.4\rightedgewidth,
   frame=off,
   style=\ssbfb\setstrut\strut,
   unknownreference=yes,
   samepage=yes]
```

The last two parameters ensure that buttons pointing to the current page show up too. The menu itself will be defined in the final text flow (discussed later) using the next macro. The last command generates the alphabet buttons (figure 8).

```
\def\indexmenu[#1]#2#3%
  {\setupalign[middle]
   \noindent\button{#3}[#1]
   \blank
   \registermenubuttons[navigation][#2]}
```

Mapping SGML onto TEX

In the previous layout definitions, we already put some structure into place. This section is dedicated to relating SGML to this structure. We will use some of the interfacing commands defined in the CONTEXT SGML module. This module acts upon a TEX file



Donald Arseneau
The TEX Hierarchy
Typesetting paragraphs of a specifie
Andrey Astrelin
Graphics for TEX: a new implementat
David Barron
Portable Documents: Why Use SGML?
Vladimir Batagelj
Combining TEX and PostScript

alphabetic entries no symbols

Figure 8

that results from the conversion by the `smgl2tex` PERL script. The TEX contains at most four different commands:

```
\begSGML[tag][var1=val1,var2=val2,...,varN=valN]
\endSGML[tag]
\sepSGML[tag]
\tokSGML{identifier}
```

These commands are direct translation of:

```
<tag var1=val1 var2=val2 ... varN=valN>
</tag>
<tag/>
&identifier;
```

In SGML setups are separated by spaces (and values can be surrounded by single or double quotes), in CONTEXT we use commas as separators.

The TEX commands are interpreted according to their definitions. Apart from some more obscure alternatives, the main definitions are given here.

```
\defineSGMLgroupedentity  [tag] [init] {actions} {actions}
\defineSGMLisolatedentity [tag] [init] {actions}
\defineSGMLseparateentity [tag] {actions}
\defineSGMLspecial        [identifier] {substitution}
```

Grouped and isolated entities are invoked by `\begSGML` but only the first one has `\endSGML`. The separate entity relates to the `\sepSGML` macro. The entities using a `\begSGML` are subjected to some basic testing on proper nesting. By providing an additional argument, definitions can be defined local to other ones, but we don't need that feature here.

When a document is not processed in the same order as it is coded, or when we want to manipulate some content first, we have to save the entries and process them manually.

```
\defineSGMLgetentity     [tag] [init]
\defineSGMLprocessentity [tag] [init]
```

The first macro defines an entry as to be stored. Once stored, the entry can be flushed (executed, expanded, or whatever term suits best) using

```
\processSGML [tag]
```

The second definition has execution built in. The associated process is defined by:

```
\defineSGMLprocess [tag] {actions}
```

One can add tokens to the stored data by:

```
\appendSGML  [tag] {tokens to append}
\prependSGML [tag] {tokens to append}
```

There are a few tests available:

```
\doifSGMLelse {tag} {when data} {when no data}
\doifSGML     {tag} {when data}
\doifnotSGML  {tag}             {when no data}
```

Parameters (initializations) can be fetched by two macros. The first one returns the variable, while the second one stores it into a macro (in this case \SomeVariable).

```
\varSGML[tag]
\setSGML\SomeVariable[tag]
```

There are some more module specific macros, but the ones described here give us enough machinery to handle the MAPS biographic data.

The whole bibliography is collected (embedded) into the bib entity. The <bib> and </bib> are mapped onto \begSGML[bib] and \endSGML[bib]. The associated TEX commands are:

```
\defineSGMLgroupedentity [bib] {\page\bgroup} {\page\egroup\endinput}
```

This means that when <bib> is encountered, we go to a new page, and when the </bib> is seen, we flush the page and stop reading the file. That way we make sure we skip all kind of comments at the end of the file.

Each <bibentry> has a few associated parameters that we set to nothing by default. Where the previous definition is executed as soon as it is seen, a <bibentry> is defined as an entity that should be saved first and executed by an associated macro to be defined later.

```
\defineSGMLprocessentity [bibentry] [type=,id=,language=]
```

The next definitions are processed directly. The br entity is the only one that is defined in the XML way:
. It comes alone.

```
\defineSGMLseparateentity [br] {\blank}
\defineSGMLgroupedentity  [tt] {\bgroup\tt} {\egroup}
\defineSGMLgroupedentity  [em] {\bgroup\em} {\egroup}

\defineSGMLgroupedentity [itemize] {\startitemize[packed]} {\stopitemize}
\defineSGMLgroupedentity [item]    {\item}                 {}
```

In an average document, we can stick to definitions like this, but here we have to manipulate data. The rest of the definitions take care of storing the data in a structured way. In a few moments we will see the related processing macros.

```
\defineSGMLgetentity[title]
\defineSGMLgetentity[booktitle]

\defineSGMLgetentity[author]
  \defineSGMLprocessentity[au]
    \defineSGMLgetentity[fn]
    \defineSGMLgetentity[prf]
    \defineSGMLgetentity[sn]

\defineSGMLgetentity[editor]

\defineSGMLprocessentity[published][volume=,pages=]
  \defineSGMLgetentity[series]
  \defineSGMLgetentity[journal]
  \defineSGMLgetentity[publisher]

\defineSGMLgetentity[keywords]
  \defineSGMLprocessentity[key][index=]

\defineSGMLgetentity[abstract]
```

Now, what happens when the file is read? The first meaningful entity is the outer <bib> one: it starts a new page. Then comes a sequence of over 700 <bibentry>'s with embedded data. Each entry is temporarily saved, and then executed. In the process, the

individual data components are saved and processed in due course. The next definition defines what should take place after a bibliography entry is read in (stored).

```
\defineSGMLprocess[bibentry]
  {\page
   \bgroup
   \setSGML\CurrentType[type]
   \setSGML\CurrentId[id]
   \setSGML\CurrentLanguage[language]
   \processSGML[bibentry]
   \processaction
     [\CurrentType]
     [inproceedings=>\HandleBook    ,
            article=>\HandleJournal,
            message=>\HandleJournal,
            verslag=>\HandleJournal]
   \egroup}
```

The macros `\setSGML` save some parameters for later use. Next we process everything between `<bibentry>` and `</bibentry>` that was automatically saved. Due to previous definitions, this means that some components are also saved for later use (like the index entries), and some are processed immediately (like `<published>`).

```
\defineSGMLprocess[published]
  {\setSGML\CurrentVolume[volume]
   \setSGML\CurrentYear[year]
   \setSGML\CurrentPages[pages]
   \setSGML\CurrentSize[size]
   \processSGML[published]}
```

We use the type of entry to determine the next action. There are some more entries, but those are simply skipped.

```
\def\HandleJournal%
  {\CheckPublication{\processSGML[journal] \CurrentVolume\space\CurrentYear}
   \HandlePublication}

\def\HandleBook%
  {\CheckPublication{\processSGML[booktitle]}
   \HandlePublication}

\let\CurrentPublication\empty

\def\CheckPublication#1%
  {\doifnot{#1}{\CurrentPublication}
     {\xdef\CurrentPublication{#1}
      \doglobal\stripSGML\CurrentPublication\to\CurrentPublicationAscii
      \publication[\CurrentPublicationAscii]{#1}
      \placelist[publicationtitle]
      \page}}
```

These macros demonstrate that when manipulating data, one cannot do without an occasional more complicated macro. Each article in the bibliography is either a MAPS one, or one published in proceedings. Although each set of articles is preceded by a description of the current volume, using this information to trigger a new volume is not straightforward. It proved to be easier to use a change in volume description as a trigger. The main reason for this is that the bibliography is rather flat, and articles are not embedded in something like `<volume>` and `</volume>`.

The second complication is that we have to make sure that cross references passed to the sectioning commands (between square brackets) are not disturbed by long sequences of low level TEX resulting from expansion. The `\stripSGML` macro makes sure we get a rather pure string.

```
\def\HandlePublication%
  {\setupinteractionmenu[right][state=stop]
```

```
\setupinteractionmenu[publication][state=start]
\setupbottomtexts[edge]
  [][{\hfill\menubutton[bottom]{view}[file(\CurrentId)]\hfill}]
\publicationtitle{\processSGML[title]}
\doifSGMLelse{abstract}
  {\processSGML[abstract]}
  {{\sl geen samenvatting (no abstract)}}
\page}
```

Each publication is typeset on a separate page. At the right we put a dedicated menu that we still have to define. In the bottom right text we put a button that brings us to the typeset article. When no abstract is given, we print a remark. The publication title is, confirming a previous set up, expanded when written to the auxiliary file.

We still have to define the publication menu, and since we now know what we are dealing with, it makes sense to do it right away. It is, by the way, not really a menu. Embedded in a bit of typographic trickery, we place the linked keyword and author lists, a button that brings us back to the relevant table of contents, and a bit of data about the article.

```
\startinteractionmenu[publication]
  \setupinteraction[color=white]
  \startalignment[middle]
  \topskipcorrection
  \ssbf\setupinterlinespace
  \processSGML[keywords]
  \vfill
  \processSGML[author]
  \vfill
  \noindent\gotobox
    {\strut\limitatetext{\CurrentPublication}{.8\hsize}{...}}
    [\CurrentPublicationAscii]
  \blank
  \strut p.~\CurrentPages\quad\CurrentSize~kB\quad\CurrentLanguage
  \stopalignment
\stopinteractionmenu
```

This leaves us with two processes: those that handle the keyword and author entities; both are made up of smaller components.

```
\defineSGMLprocess[au]
  {\bgroup
   \setSGML\CurrentIndex[index]
   \processSGML[au]
   \doifSGML{fn}{\appendSGML[fn]{\space}}
   \doifSGML{prf}{\appendSGML[prf]{\space}}
   \doglobal\stripSGML\processSGML[title]\to\CurrentTitleAscii
   \noindent\hbox to \hsize
     {\strut
      \hss
      \titlebyauthor
        [&\CurrentIndex&\CurrentTitleAscii]
        {\processSGML[fn]\processSGML[prf]\processSGML[sn]&\processSGML[title]}%
      \coupledauthor
        [\CurrentIndex]
        {\processSGML[fn]\processSGML[prf]\processSGML[sn]}%
      \hss}
   \par
   \egroup}

\defineSGMLprocess[key]
  {\bgroup
   \setSGML\CurrentIndex[index]
   \doifelse{\CurrentIndex}{}
     {\doglobal\stripSGML\processSGML[key]\to\CurrentKeyAscii}
```

```
    {\doglobal\stripSGML\CurrentIndex\to\CurrentKeyAscii}
  \noindent\hbox to \hsize
    {\strut\hss\coupledkeyword[&\CurrentKeyAscii]{\processSGML[key]}\hss}
  \par
  \egroup}
```

The main macros here are the `\coupled...` ones. These became available when we defined the related linked registers. In ConTeXt an index entry looks like:

```
\index{alpha} \index{alpha+beta}
```

The second example defined `beta` as a subentry under `alpha`. Of course we want to see a real α and β, so in practice we separate the typeset entry and sort key, like in:

```
\index[alpha+beta]{\mathematics{\alpha}+\mathematics{beta}}
```

When an entry itself contains a `+`, one can alternatively use an `&`, and by (optionally) passing it as the first character, one can signal the index sort script that this character is to be considered a separator. By doing so we make sure a `+` is not misinterpreted as a separator, as it happens that the `+` is sometimes part of the text.

```
\coupledauthor
  [\CurrentIndex]
  {\processSGML[fn]\processSGML[prf]\processSGML[sn]}
```

This sequence typesets the author's name (composed from three parts) and passes the sort index as provided by the database.

The main text

Now that everything is set up, we can safely start with the main text flow, the least interesting part of the setup. To save some space, I will skip the title page and colofon. The occasional change in layout ensures that the menu is extended to the bottom of the screen.

```
\starttext

\setuplayout[bottom=0pt]
  \startstandardmakeup
    title page
  \stopstandardmakeup
\setuplayout[bottom=20pt]

\structuralelement[publications]{Publications}
  \startcolumns[n=3]
    \placelist[publication][criterium=all]
  \stopcolumns
  \input mapsbib.tex

\structuralelement[keywords]{Keywords}
  \setupinteractionmenu[navigation][state=start]
  \setupinteractionmenu[right][state=stop]
  \startinteractionmenu[navigation]
    \indexmenu[keywords]{keyword}{Keywords}
  \stopinteractionmenu
  \placeregister[keyword]

\structuralelement[authors]{Authors}
  \startinteractionmenu[navigation]
    \indexmenu[authors]{author}{Authors}
  \stopinteractionmenu
  \placeregister[author]

\structuralelement[titles]{Titles}
  \startinteractionmenu[navigation]
    \indexmenu[titles]{titlebyauthor}{Titles}
```

```
  \stopinteractionmenu
  \placeregister[titlebyauthor]

\structuralelement[ntg]{NTG}
  \setuplayout[bottom=0pt]
  \setupinteractionmenu[navigation][state=stop]
  \setupinteractionmenu[right][state=start]

  The main ... at: \goto {www.ntg.nl} [URL(ntg)].

\stoptext
```

The URL is defined previously by: `\useURL[ntg][http://www.ntg.nl]`.

Fine points

When typesetting a large quantity of data automatically, one must make sure the outcome looks acceptable. The fact that TEX can typeset so well, does not automatically mean that its output always looks correct. When designing a layout, most of the time goes into the fine points. Setting up a basic page frame and defining some basic structure is a matter of minutes, but finding the optimal and pleasing dimensions sometimes takes hours. In the MAPS bibliography quite some time went into checking the input and making the index consistent within the boundary conditions. This document also provided an ideal environment to play a bit with SGML processing.

To handle unforeseen overfull boxes, two methods are used. First we have made sure TEX's emergency pass takes care of bad line breaks:

```
\setuptolerance [verytolerant]
```

The other method concerns automatically limiting the length of index entries and titles to about 80% of the available width. Most of this is handled by the higher level macros, but on one place we have to do it manually, saying:

```
\limitatetext {\CurrentPublication} {.8\hsize} {...}
```

Processing

Processing an XML file comes down to converting the file into TEX and then processing that file. In most cases, the SGML file is checked beforehand, but one can never be sure of that. Therefore, much of the low level SGML processing macros is dedicated to checking symmetric use of entities and nesting. These features were added when we were given some 'professional' SGML databases, which unfortunately, and in spite of all parsers, had lots of dangling `<this>` and `</that>`'s. For processing the well coded MAPS database, we could have done without all this overhead.

Expansion

I have already mentioned expansion a few times. In this section I will spend some more words on this topic, if only because one needs to be well aware of this TEX feature when defining an interface to SGML.

When one writes an article, the typeset text, apart from floats, the table of contents and the index, show up at the place where they are defined. In the more complicated documents, think of educational materials, the order can be different from the order in which the document is coded. Questions and answers can be coded along side each other, but may show up in different places. Formulas may be repeated in an appendix. The MAPS bibliography is simple in the sense that information is only presented once, but moderately complex because the order differs from the order in which the information is coded. Some data, like the indices, are not coded at all but constructed from pieces of code.

Definitions, formulas, chapters, and the like are often numbered. The numbers are not coded, but generated by the TEX macro package that is used. In the bibliography section, numbers will not be typeset. What is not visible in the resulting document is that there are more entries in the XML file than those that show up in the PDF file. It proved to be more practical to distill volume specific information from the entries describing the articles, than to use the entries (one per volume) dedicated to that purpose.

What are the consequences of this for processing the document? The reordering forces us to save the data temporarily, which in itself is not that complicated. A first version used the CONTEXT buffering system. This means that an entry was saved and then reprocessed under different conditions. There was a separate pass for constructing the index entries, making up the abstract, etc. The current implementation uses the save and recall facilities as provided by the CONTEXT SGML module. This is slightly faster and proves to be more convenient.

When we are manipulating data in TEX, we undoubtly come across expansion. In TEX, tables of contents, indices, cross references, certain optimizations and more depend on processing the file in several passes. The current pass uses the information saved during a previous pass. When dealing with tables of contents and indices, there are two extremes: the data is fully expanded versus the data is not expanded at all. In CONTEXT, by default, the content is not expanded. This means that no complicated protection mechanism is needed to prevent unwanted expansion.

But, as said, when processing SGML, we save the data in order to be able to manipulate it. When we are done with these manipulations and pass the data to a section title that also has to show up in a table of contents, expansion definitely *is* needed, but only as far as is comfortable. I will illustrate this with an example. Consider that we have defined:

```
\def\SomeTitle{Some title}
\def\Some     {Some}
\def\Title    {title}
```

When these definitions are permanent, it does not matter how they are saved in an external file that is read in during the next phase, so all next alternatives work out okay.

```
\chapter{Some title}
\chapter{\SomeTitle}
\chapter{\Some\space\Title}
```

When, however, the definition of `\SomeTitle` changes halfway the document, or over 700 times in a bibliography as discussed here, we're in trouble. In that case, we have to expand `\SomeTitle` prior to writing it to the auxiliary file. In a simple case like this, full expansion is no problem, because we are dealing with pure text. In the document at hand however, we have definitions like:

```
\def\Title{The use of \begSGML[tt]verbatim\endSGML[tt] in \tokSGML{tex}}
```

When we want this to end up in the auxiliary files in this way, we can use a one level expansion, which means replacing the macro `\Title` with its content. Unfortunately, macros like `\Title` can be nested or buried inside others, so actually we will need expansion until nothing can be expanded any more. This in itself is not that complicated, but one needs to be aware of this feature and potential side effects in order to write the right interface between TEX and SGML.

Actually, this expansion trickery is the only thing that complicates SGML processing as discussed here, but once one understands the problem, one can use the appropriate CONTEXT features to sort it out. In most cases the option `expansion=yes` will do the job: expand everything as far as permitted.

Conclusion

The document described here can be found at the ConTeXt part of our internet site www.pragma-ade.nl, and, of course, at the NTG site www.ntg.nl. The complete MAPS archive will be put on CDROM in PDF format along with the bibliographies in HTML and PDF format.

In this article I have tried to give an impression on how such a document is defined. As with many interactive documents that involve some manipulation of the content, there is an easy to follow definition part as well as a more complicated one dealing with fine tuning and manipulations. One can do wonderful things with TeX, but the more one wants, the more one ends up in dealing with typographic issues and low level programming. ConTeXt provides the user with enough structural mechanics to ease at least part of the task. Defining the layout, registers and lists is not that hard, but the SGML part doesn't always deserve a beauty prize. Even if not all of what is presented here is clear, at least it gives an impression of what kind of trickery is needed. I leave it up to the reader to imagine what nasty tricks would be needed if the MAPS articles themselves were to be coded in SGML.

One of the best things about TeX is that often one has to define the layout and manipulation macros only once. This is what distinguishes TeX from other systems, but at the same time it forces the user to develop some skills in typographic programming. Fortunately, the results are rewarding.

software

4TEX 5.0 for Windows and the 4allTEX CDroms

Wietse Dol
Erik Frambach

abstract

Every year many major software distributors launch a new version of their software products. They all tell you that there are fundamental improvements and bug fixes and that you have to upgrade immediately. With the release of edition 5 of 4allTEX you could ask yourself what this new edition offers as compared to e.g. the older versions of 4allTEX and the TEX-live CDrom. Below we will give a summary of all the goodies of the new 4allTEX distribution and give you pointers to the internet where you can find more information.

Introduction

The first question that people ask is why we are always talking about 4TEX and 4allTEX. What's the difference? The answer is perhaps subtle but there is indeed a difference. 4TEX is the user interface that we have built to set up a complete workbench for TEX and its many friends. Using 4TEX you can edit, view, print documents, convert graphics etc. etc. 4allTEX is the name of the two CDroms and it symbolizes the fact that not only people who use the 4TEX workbench can benefit from the CDroms. Indeed, everybody (that's why we speak of forallTEX) can use the programs, documentation etc. without using 4TEX. Just speaking as a mathematician: 4TEX is a subset of 4allTEX.

The previous versions of 4allTEX all supported the MS-Dos operating system (with emTEX as the TEX 'engine'). Now a lot of people are asking for a true Windows implementation. Version 4 of 4TEX had some options to use Windows programs but still was MS-Dos based. Windows 95/98 and NT are all 32-bit operating systems and people asked us to think about a true 32-bit 4TEX workbench. Since the core of 4TEX was written in the 4Dos batch-language a 32-bit 4TEX simply means a complete rewrite of it.

So we started to make a completely new version of 4TEX with no backwards compatibility: the new version will not work on an machines running MS-Dos or Windows 3.x. We didn't throw away everything though: the 4TEX philosophy still applies. The fundamental principle of 4TEX has always been not to force anyone to use any particular program. We simply gave you a tool with programs we use in a production environment that we think are useful to you too. At any time, anywhere within 4TEX you can change or add the programs you prefer to use. This means that 4TEX should be extendable/changeable in almost every respect. This was done in the old version in a file that kept the MS-Dos environment variables settings. We didn't start to write our own editor program and give it certain buttons to add TEX functionality for two reasons. The first one is that there are many splendid editor programs already available and the second reason is that you probably have different ideas about what a good editor program is. So 4TEX should be a program (or to use the buzz word: an agent) that could be an intermediate between editor program, TEX programs, and all other tools that you need to publish documents on paper, the internet, etc.

4TEX version 5 was written in Borland Delphi 4.0 and it uses many ASCII configuration files that can be changed/updated in order to customize 4TEX to your taste. Just one example: the 4TEX menus "speak" 9 different languages that can be changed on the fly (English, German, Dutch, French, Polish, Danish, Czech, Slovak, and Russian). Just by copying the screen text file and translating it you can add your own language. If you don't like the text on a button or somewhere else you can change the language dependent screen file and have your own text displayed. This is all done by using any ASCII editor program, no programming experience required. Adding programs, printers, viewers, formats etc. is done by changing a certain ASCII file. Naturally the program works smoothly in a network environment and there is extensive online help available. 4TEX is a true Windows program and acts like most Windows programs: it saves the screen positions, the screen sizes, font sizes, and many other settings that can be changed while running 4TEX.

Making 4TEX for Windows took about 1.5 year and during this time it was tested by many beta testers, whom we can never thank enough. It was used at the faculty of Economics of the University of Groningen as a production tool for articles, books etc. and as the ultimate test was used it to write the 550-page 4TEX book. This book is a nightmare in complexity as it stretches TEX and its friends to their limits. It works, and we are now confident that a larger audience can use it. That's why we are now releasing 4allTEX edition 5. Below you will find a summary of the functionality and features of 4allTEX. We hope that you are convinced that 4TEX is worth trying.

The 4TEX manual

Making 4TEX for Windows we decided not only to completely rewrite the program, but also decided to write a new book. The 160+ page booklet that came with the old 4TEX version wasn't enough, we thought. What we would like to write is a book that tells you about the good and bad points of TEX. That tells you how the Windows program 4TEX can be used, how it can be changed to suit your needs. That tells you everything about the Web2C TEX implementation and all its parameter settings. That gives you an overview of the TEX dialects and a detailed introduction to plain TEX, LATEX, and ConTEXt.

The reason is simple. You only need this book plus the 4allTEX CDroms to have everything you need to start writing your own TEX documents. If you like TEX you will buy more books and become an expert, if you stop using TEX (such people do exist) you didn't waste much money – 15 Euro isn't that much and on the CDroms there are many non-TEX programs that are extremely useful. The book isn't only for novice TEX users. We believe experts can learn from it as well. E.g., this is the first book that describes Web2C in detail. Indeed 4all!

As an appetizer we will present the table of contents of the book so you'll get an idea of what you would get:

The 4TEX book is sold separately from the 4allTEX CDroms. This has a legal reason. There are some (extremely useful) programs that forbid distribution by inclusion of CDroms in a book. Selling them separately has some advantages: it means you can buy the CDroms without manual. The 4TEX book is available on the CDrom in PDF format anyway, so you can read it online and decide whether you want to buy it later.

The Setup, 4TEX, 4Spell, and 4Project the programs

The programs 4Project and 4Spell were already discussed in MAPS 22. These programs are developed in such a way that they can be used stand-alone. In this article we will have a closer look at the Setup program and the 4TEX workbench.

All decent Windows programs come with an installation (and uninstallation) program. This program will copy the files from CDrom to a user specified directory, create the necessary registry settings, make the shortcuts (icons) on the desktop etc. 4TEX has such a clever installation program. If you have not installed 4TEX yet (the Windows registry will tell) then the Setup program will start automatically. Or if you want you can start the Setup program manually. The Setup program will display a screen as shown in Figure 1.

The Setup program offers you two ways of installing 4TEX: a simple and an advanced installation. Basically the simple setup will run completely from CDrom or install everything on your hard disk (see Figure 2). This simple installation can be done in less than a minute if you decide to run 4TEX almost completely from CDrom. It will only create the shortcuts and the `4TEX.INI` file (the Windows equivalent of the environment files `texuser.set`, `system.set` from the MS-Dos days). The `4TEX.INI` file contains all the personal settings for your workbench. This approach is useful to people with limited hard disk space or people who just want to try out TEX and 4TEX. If you don't

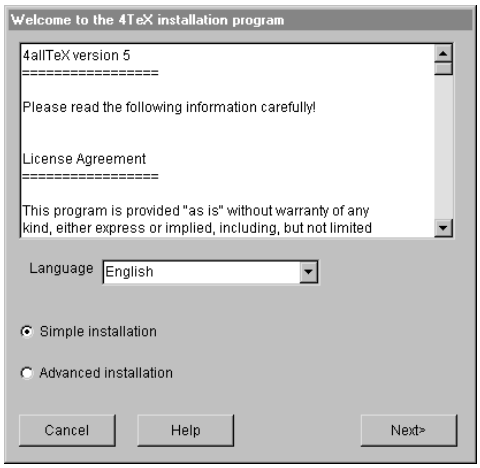


Figure 1. The opening screen of the 4TEX setup program

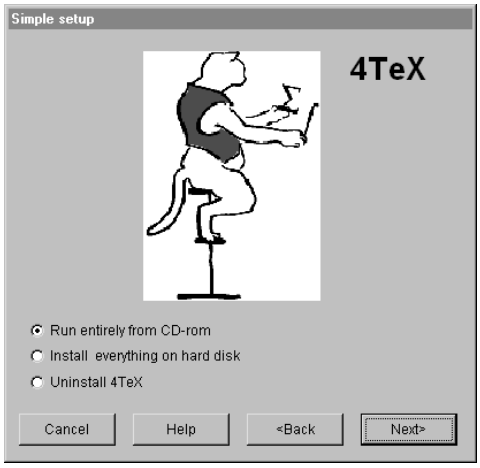


Figure 2. The Simple installation program

like 4TEX, uninstalling everything is even simpler than installing.

The advanced installation offers a menu from which you can specify which modules you want to install. Modules are parts of the CDrom that have a certain functionality, see Figure 4. The modules are plain ASCII files and are an easy and flexible way to offer people all kind of user specific TEX installations. It also makes it possible that when people ask for extra installation support (write to `4TeX-support@ntg.nl`) we can put extra installation modules on the internet and you can download them and use them. Advanced installation is for people who know about TEX and its friends and want a tailor made TEX system.

The 4TEX workbench gives you a user friendly interface with a large set of utilities. It has a complete Windows look and feel, but some hard programming was necessary to get all the programs to work together. For instance: all

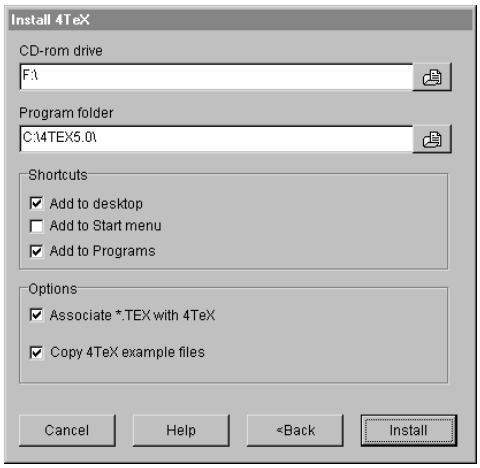


Figure 3. Choose shortcuts and association of `.tex` files to 4TEX

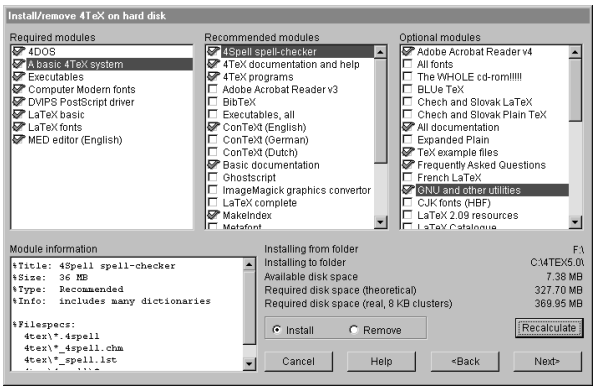


Figure 4. The advanced installation program offers you to install modules

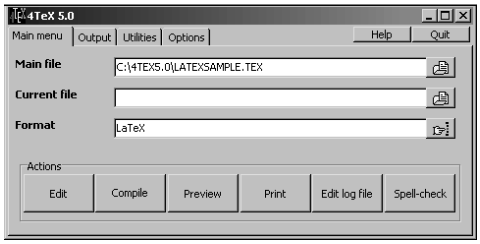


Figure 5. The 4TEX workbench

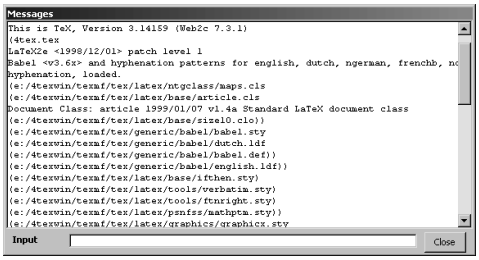


Figure 6. The redirection window

the TEX programs are 32-bit console applications that write messages in "MS-Dos" style. 4TEX redirects this output to a window and makes it possible to scroll through the output (see Figure 6).

Since the "DOS-boxes" are still being used and since we want to give users the possibility to do their own programming, the shareware program 4DOS from JP-Software is still the best alternative for `COMMAND.COM` or `CMD.EXE`. So we arranged a deal with JP-Software: the 4allTEX CDroms come with a licensed run-time version of 4DOS and 4NT. The same holds for the MED editor. This means that all the basic functionality that 4TEX offers doesn't require you to register any shareware anymore. Of course we still offer you some extra tools that are not licensed, but those programs aren't essential for the 4TEX workbench.

Almost all functionality available in the older 4TEX versions is still there: block compilation, BibTEX, MakeIndex, MetaPost, Metafont, conversion tools, etc., but we added some additional features, as well. E.g., you can now convert almost any graphic picture using the ImageMagick software and the graphics conversion menu. Spell-checking is done by 4Spell, and if you need some help you can read many on-line documents or start some wizards.

4TEX could have worked with emTEX or MikTEX but we decided to use fpTEX, the fast, well-known Web2c distribution for Win32. We chose Web2c because it is the most widely used TEX implementation world wide that runs on UNIX, Windows, DOS, VMS, Atari and several other platforms. We could discuss 4TEX for another 200 pages but all of that you can read in the 4TEX book, or find it on the internet: `http://4TeX.ntg.nl`. Just to show you the fun of 4TEX: instead of using the large main menu you can use the small yet fully functional toolbar. Putting it on top of your editor it looks as if the 4TEX buttons have become part of the editor program (see Figure 7).

4allTEX CDroms

If you want to see what 4allTEX offers you can find complete file listings on the 4TEX internet site, `4TeX.ntg.nl`. If you need help you can contact the 4TEX discussion list or write to the 4TEX-support team at `4TeX-support@ntg.nl`.

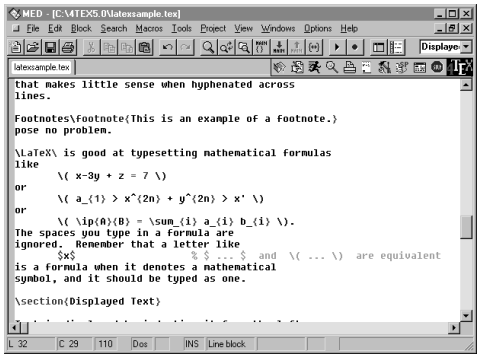


Figure 7. The 4TEX workbench toolbar as part of the editor

If you are looking for updates or a list of Frequently Asked Questions, just visit the internet site. We will conclude this article with a summary of 4allTEX's main features and we hope that you will tell us what you like, what you can contribute to 4TEX and what we need to improve.

- contains the latest 4TEX version, 5.0. 4TEX gives the user a friendly interface with a large set of utilities.
- contains many TEX and Window software in direct executable form.
- supports many different TEX formats, such as plain TEX, LATEX 2_ε, MeX, TEXinfo, LATEX 2.09, ConTEXt, CSLATEX, pdfTEX, ε-TEX, FRLATEX
- supports many printer drivers (including color deskjets, linotronics, matrix printers, 300/600/1200 dpi laserprinters, laserjets, fax, ghostscript, etc.
- 4TEX can output PDF directly.
- 4TEX supports graphics: both for previewing and printing. All functionality of e.g. GhostScript, LATEXcad, Gnuplot, Mayura Draw, Paint Shop Pro, is included, ready to use.
- includes 4Spell: a spell-checker for languages English (UK), English (USA), Dutch, German (old and new spelling), French, Spanish, South-African, Italian, Polish, Chech, Slovak, Russian, Swedish, Danish, Slovenian.
- contains conversion utilities: WP to LATEX/TEX, MS-Word/Chiwriter, PC-Write/troff to LATEX, deTEX, midi2TEX, (La)TEX to HTML, LATEX to RTF, etc.
- 4TEX can convert almost any graphics format, using ImageMagick software.
- contains a huge set of .sty .cls .mf .pk .tfm .bst etc. files.
- 4TEX supports Metafont with automatic font generation.
- 4TEX supports MetaPost for drawing pictures.
- 4TEX fully supports PostScript fonts. E.g., with automatic font generation using GSFTOPK.
- With 4TEX you can view and print directly in e.g. Times font on any printer.

- 4TEX contains database utilities, supports MakeIndex, etc.
- 4allTEX comes with a choice of fine editors, such as MED, PFE, WinEdt, NoteTab, UltraEdit and Xemacs.
- 4TEX supports *block compilation*: mark a small part of your (large) document, and only that part will be compiled and previewed.
- 4allTEX contains a huge set of documentation, including TEX/LATEX/Metafont/MetaPost documentation and tutorials both for novices and gurus, including:
  - MAPS issues (both Dutch and English articles; starting in 1988; more than 2000 pages with high density information arranged in a large set of PDF files),
  - all mail from the TEX-NL and the TEX-D-L mailing list,
  - all BASKERVILLE issues from UKTUG,
  - A lot of tutorials in TEX/LATEXsource(also in .dvi, .pdf, and .ps files),
  - The 'TEX book' and the 'Metafont book' in TEX source format,
  - FAQ about TEX, PostScript, fonts, etc.
- 4allTEX contains specials: chess (including Chinese), bridge, music, crossword, go, and more; including all fonts.
- 4allTEX contains Arabic, French, Cyrillic, Polish and Chinese packages.
- With 4TEX it is very easy to generate completely new format files.
- 4allTEX contains a lot of DVI and PostScript utilities.
- 4allTeX contains extensive bibliographies on TEX-related topics.
- Because TEX is extremely useful for the generation of HTML, PDF and other multimedia documents you will find a lot of those programs on the CDrom. Lots of documentation is available in HTML format.

TEXniques

Toolbox: the toolbox?

Maarten Gelderman

abstract

This MAPS is about the TEX Toolbox, about other programs than TEX itself. So this MAPS's toolbox should probably deal with this kind of material. As a consequence this toolbox is even more eclectic than earlier ones. First I will show you how I make mailings to NTG-members, by combining Excel and LATEX. Next I will present the most ugly regular expression I know of, and finally I will say something about using makefiles.

keywords

mail-merge, regular expressions, emacs, Excel, makefiles

Using Excel and LATEX to do a mail-merge

A recurring question on forums like TEXNL is whether it is possible to use LATEX to do mail merging. Of course the answer is yes, and numerous ways are available. One might create a single large TEX file by using the reporting facilities of a database program like Access. However, as 99% of the resulting document will consist of mere repetitions of the body text, this approach is rather inefficient. Table 1 shows a solution that avoids repetitions. The trick is to define a new command (`\myletter`) which contains the text of the letter, the `letter`-environment and the `\opening` and `\closing` statements. In the body of our document, we now repeatedly use the `\myletter` macro with appropriate arguments to produce the individual letters.[1]

The `\myletter` macros can be generated by a database application, by a Perl script, of even by using a simple spreadsheet-formula. Of course it is more elegant to produce the file directly with Perl or the database manager, however, for most applications using a spreadsheet suffices. Elementary functions for sorting the database and making selections are available. The only thing we have to do is to find a way to put the required fields of the database in a single cell for each row. This also is easily accomplished. In a single cell, we enter a formula similar to that presented underneath:

```
=CONCATENATE("\myletter{";+IF(K2<>"";\
+CONCATENATE("Dear ";K2;",");\
"Dear member,");"}{";J2;"}{";I2;"}\
{";L2;"}{";M2;"}{";+IF(O2<>"";\
O2;N2);"}{";P2;" ";Q2;"}{";R2;"}")
```

Table 1. Mail merge using the standard LATEX letter-class

```
\documentclass{letter}
\newcommand{\myletter}[8]{%
  \begin{letter}{#2 #3\\#4 #5\\#6\\#7\\#8}
    \opening{#1}

    This is a letter. A rather short one, but a
    letter nevertheless.

    \closing{With kind regards}
  \end{letter}}

\begin{document}
\myletter{Dear Karel,}{J.F.}{Krammers}
      {University of Nowhere}{%
         Department of Improbable research}
      {Piet Heinstraat 10}{1399 EW Muiderberg}{%
         Nederland}
\myletter{Dear Jan,}{J.H.}{Drupnats}
      {Ministry of Silly Walks}{}
      {Binnenhof 30}{2222 KH Den Haag}{Nederland}
\end{document}
```

This formula (the backslashes at the end of the lines just indicate that it should be put in a single cell) concatenates a number of text fields. Everything between quotation marks is put into the cell verbatim. The letter number-combinations point to the cells containing the data. The K-column, for instance contains the first name of the addressee. The if-clause checks whether a first name is present in the database. If this is the case it puts 'Dear firstname,' in the first parameter field of the `\myletter`-command, if it is not present, 'Dear member,' will be used instead. The remainder of the formula refers to the other fields of the database in a similar way.

After selecting the appropriate records, we just use copy-and-paste to put the results of the concatenation formula into the TEX-file and are ready to generate our document.

An unreadable regular expression

If a produce-the-least-readable-regular-expression-contest would exist, the next one probably would have a good

1. In terms of computation time required this approach is still rather inefficient: the bodytext has to be typeset by TEX for each individual letter. It is possible to solve this problem, but that would probably cost more time than we would ever save by this improvement.

chance of winning it:
`\\\([’\|\^\|‘\|"]\)\([^\{]\)` → `\\\1{\2}`
Perhaps the most surprising thing about this regular expression is that it is actually useful. Although it admittedly is prone to typing errors, it is even easy to understand.

What does this regular expression accomplish? It replaces every occurrence of an accent directly followed by a character, with the same accent, followed by the same character, but after replacing the character is placed between curly braces. A small example: `\’a` becomes `\’{a}`, `\^e` becomes `\^{e}`, and so on.[2] How do we accomplish this replacement? First we have to decide which program we want to use. I decided to use emacs, as I do all my editing in this program, but one might decide to use any other program that is able to deal with regular expressions. Perl would be another likely candidate. Now we have to build the regular expression we want to use for searching. This also is fairly easy. The first character we are looking for is the backslash. Unfortunately this is a character with a special meaning, hence we have to escape it. To indicate that we are looking for a single backslash, we start the regular expression with `\\`. The next character we will be looking for is the accent. We want to treat this accent slightly different from the backslash: it has to be stored as we will need it in the replacement operation. In order to store it, we start a new group. This is done with the next two characters in the regular expression: `\(`. A few characters later, the group will be closed with `\)`. The sequence between the opening and the closing of the group will be used to find the accent and store it. As this is our first group, it will be saved in `\1`.

The group itself contains the different accents. The accents are separated by the ‘or’-operator: `\|` and all alternatives are places between bracketed braces (`[]`), the order in which the accents themselves appear is irrelevant. The `’` is found by the first character in the group, which is identical to `’` itself, `^` has a special meaning in regular expressions, so it is escaped and will be found by `\^`, `‘` and `"`, don’t have special meanings and can be used directly.

After matching the accent we open a new group with `\(`, which is closed by the `\)` at the end of the regular expression and the contents of which will be stored in `\2`. This group is used to find the character to be put between braces. The character itself is defined in a somewhat peculiar way. Instead of listing all possible characters, we define this group as every thing that is *not* a curly brace (if we wouldn’t do this `\’{a}` would be replaced by `\’{{}a}`, which it not wrong, but not particularly desirable either). The operator `^` is the not-operator, and the curly brace itself is escaped by putting a backslash in front of it.

The search regular expression has been constructed by now. The replacement expression if fairly short. First we put the backslash in place again (`\\`), next we put the accent (which has been stored in register 1) in the replacement text by the command `\1`, next we place the opening curly brace (which this time does not have to be escaped: {), the character itself (which is stored in `\2`) and the closing curly brace (}) and we are done.

Makefiles

If you already know something about makefiles, you can skip this section. I know hardly anything about this topic. Until recently my conviction was that makefiles were made by other people who had written a program I wanted to install, and the only thing I ever did with a makefile was to change some site specific settings in the first few lines of it. As those latter modifications lead to errors every now and then, I decided it might be worthwhile to read the manpages. It turned out they didn’t exist. A manpage for the make-program did exist, but did not contain any information about makefiles themselves. Fortunately, it did refer to a file in my doc-directory. As this file did not exist either, I decided to follow a somewhat unusual approach: I consulted a real manual. Here I learned that makefiles, although you can probably do very complicated things with them, are in fact really simple. The only thing you have to remember is that you have to use tabs for indenting and not spaces.

Let’s examine a very simple makefile:

```
doc.dvi: doc.tex
      latex doc
      bibtex doc
      latex doc
      latex doc
```

Make sure that on line 2–5 tabs instead of spaces are used for indenting. We save this file and give it the name Makefile (with a capital M, if you choose another name, you have to use `make -f file` instead of just `make`). After saving the makefile, we can compile our document by simply entering the command `make` from the command line. Make checks whether doc.tex is newer than doc.dvi and if this is the case, it runs the four commands on line 2–5.[3]

The example given above, although useful, is not very inspiring.[4] A slightly more complex example is given below:

```
all: dea.dvi dea.1
        dvips  -f dea.dvi -o print.ps
```

2. The only reason why this operation is useful, is that latex2rtf cannot deal with accented characters without the curly braces.

3. If you want make to run those four commands any time you enter make, just replace `doc.dvi` by some other name, e.g., `dvi`. The file `dvi` does not exist and will never exist. Consequently make will always run the four commands.

4. It is not very efficient either, each line is executed in a separate shell, it would be more efficient to put the four commands on one line separated by semicolons.

```
dea.dvi: dea.tex dea.1
        latex dea
        latex dea

dea.1: dea.mp
        export TEX=latex; mpost dea

clean:
        rm *~ *.dvi *.log *.?lg
```

If we just type `make` (or `make all`), make will first check whether the files mentioned after `all:` are up-to-date. If not, that is for instance if `dea.tex` or `dea.1` has been modified more recently than `dea.dvi`, first the commands for those files are run and next the command for all is run. We could also run those commands separately, by entering `make dea.1`, make would only check whether `dea.1` is up-to-date and run the required commands if necessary. Another interesting featue is the line `export TEX=latex; mpost dea`. First we tell our shell that LaTeX should be invoked to process TeX-commands issued by MP, and next we run MP. Because each line has its own shell, this setting does not carry over to other commands (and consequently, we cannot put them on separate lines). Makefiles can be made more complex if desired. It is, for instance, possible to use parameters to make a Makefile more generic (in this example I also added two comment lines):

```
# Begin definitions
FILE=dea

# Begin dependencies
all: $(FILE).dvi $(FILE).1
        dvips  -f $(FILE).dvi -o print.ps

$(FILE).dvi: $(FILE).tex $(FILE).mp
        latex $(FILE)
        latex $(FILE)

$(FILE).1: $(FILE).mp
        export TEX=latex; mpost $(FILE)
```

This example should be clear withour further explanation. One final remark, which will also end this toolbox, is in order however. The variables declared can be used by prepending them with a dollar sign. In order to use the dollar sign itself, it has to be escaped by another dollar sign. The next excerpt from a Makefile demonstrates this latter feature.

```
fonts:
        tex pad

tfm:
        for f in *.pl; do pltotf $$f; done
        for f in *.vpl; do vptovf $$f; done
```

Paul Huygen

TEXniques

Maak een logo met behulp van literate programming

abstract
In dit artikel wordt beschreven hoe de *literate programming* techniek gebruikt kan worden om complexe taken uit te voeren en te documenteren. Als voorbeeld wordt een logo ontworpen met behulp van de gereedschappen AWK, picTEX en LATEX.

1 Inleiding

Dit artikel is eigenlijk een testimonium paupertatis. Het laat zien hoe iemand die te lui is om Metapost te leren, toch een logo kan maken. Inplaats van vijf regels Metapost code moet er dan een veel ingewikkelder programma komen. Gelukkig kan zo een programma goed gedocumenteerd worden door de *literate programming* techniek te gebruiken.

1.1 Literate Programming *Literate programming* is gebaseerd op een idee van Donald Knuth [3]. Het basis idee is, om bij het programmeren niet uit te gaan van de computer en de computertaal, maar van mensen en mensentaal. In een document legt de programmeur precies uit wat de computer moet doen, en *en passant* voert zij de code er aan toe. Praktisch komt het er op neer dat de programmeur een elektronisch document creëert, waarin stukken programma en uitleg voor de lezer elkaar afwisselen. Een stuk code heet een macro, en heeft een naam (label). In een macro kan een andere macro, die op een heel andere plaats in het document staat, ingevoegd worden. Dit gebeurt door het label van de andere macro tussen de code in te voegen. Hierdoor hoeft de programmeur zich niet te houden aan de volgorde van de code, zoals die door de programmeertaal is vereist. Bovendien is het eenvoudig geworden om top-down te programmeren. Code die bedoeld is om een bepaald deelprobleem op te lossen wordt in een aparte macro gezet met een label dat duidelijk aangeeft wat de code moet doen. Om dit idee toe te passen ontwikkelde Knuth WEB. WEB bestaat uit twee programma's. Het programma WEAVE zet een elektronisch document om in TEX code voor menselijke lezers, en het programma TANGLE zet het document om in Pascal code voor de computer. De bekendste toepassing van WEB is TEX zelf. TEX is het enige programma dat ik ken, dat in boekvorm is uitgegeven [4].

Naderhand zijn er verschillende alternatieve systemen ontwikkeld voor literate programming, voornamelijk omdat gebruikers het systeem voor andere programmeertalen dan Pascal wilden toepassen. Ieder systeem heeft zijn eigen voor- en nadelen. Belangrijke eigenschappen zijn:

- Geschiktheid voor één bepaalde programmeertaal, of voor verscheidene programmeertalen.
- "pretty printing" van de code in de macro's. De keerzijde van pretty printing is meestal, dat het programma met slechts één programmeertaal kan werken.
- Geschiktheid voor één bepaald zetsysteem of tekstverwerker of onafhankelijkheid daarvan. Verreweg de meeste systemen zijn gemaakt voor TEX of een TEX dialect zoals LATEX.
- Geschiktheid om meerdere bestanden met code aan te maken. Het oorspronkelijke

WEB programma zette alle code in één Pascal bestand. Bij de taal C bijvoorbeeld moeten er doorgaans echter meerdere bestanden aangemaakt worden, bijvoorbeeld voor de compiler en voor de precompiler. Ook is het handig als bijvoorbeeld de Makefile in het elektronische document kan worden opgenomen.

- Automatisch aanmaken van indexen. Een systeem dat voor één bepaalde programmeertaal is gemaakt, kan vaak automatisch *identifiers* herkennen en deze in een index opnemen.
- Macro's met argumenten. Het is handig als je met macro's een soort functies kunt maken met argumenten.

Een beschrijving of opsomming van de verschillende systemen valt buiten het bereik van dit artikel. Veel informatie is te vinden in de Literate Programming FAQ (`http://shelob.ce.ttu.edu/daves/lpfaq.txt`)en op `http://www.desy.de/user/projects/LitProg/Start.html`.

Voor de toepassing in dit artikel is Nuweb gebruikt. Nuweb is geschreven door Preston Briggs. Het is te vinden in het CTAN archive in directory `/web/nuweb`. Nuweb heeft de volgende voordelen:

- Bijzonder eenvoudig. Alles kan gedaan worden met een handvol commando's. Door de eenvoud is de vertaling erg snel.
- Kan voor alle programmeertalen gebruikt worden en kan verschillende programma code bestanden voor verschillende programmeertalen tegelijk genereren.
- Semi-automatisch genereren van een index. De programmeur geeft éénmaal aan dat een bepaald woord een identifier is die in de index moet, waarna Nuweb zelf opzoekt in welke macro's de identifier gebruikt wordt.
- De gebruiker formatteert de bestanden met programmeercode zelf. Dit is bijvoorbeeld handig voor debuggen.

Het programma heeft ook nadelen (die voor mij dus niet doorslaggevend zijn):

- Geen *pretty printing*.
- Geen macro's met argumenten.
- Afhankelijk van één typesetter (gebruikt LaTeX voor typesetting)

Het bronbestand voor Nuweb lijkt sterk op een LaTeX bestand. Binnen het bestand kan een instructie gegeven worden om een programma code bestand te openen. In het opgemaakte document ziet dat er bijvoorbeeld zó uit:

```
"xample.pas" 1a ≡
      Progam xample;
        var ⟨variabelen voor xample 1b⟩;
        begin
          ⟨open het invoer bestand 1c⟩
          ⟨Lees het bestand en verwerk de gegevens 1d⟩
          ⟨Schrijf het resultaat op het beeldscherm 1e⟩
        end.
      ◇
```

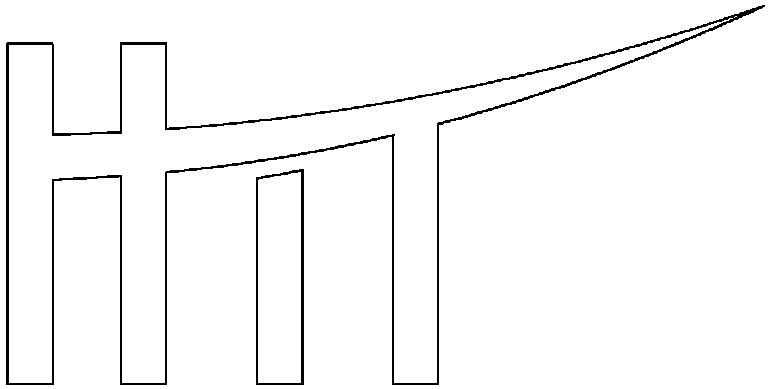
In dit codefragment wordt het bestand `xample.pas` geopend, en gevuld met programmacode. De programmacode bevat verwijzingen naar macro's die geëxpandeerd moeten worden (dat wil zeggen, dat de verwijzing vervangen moet worden door de code die in de macro staat). De macro kan op een heel andere plaats gedefinieerd worden. Dat ziet er bijvoorbeeld zo uit:

⟨variabelen voor xample 1b⟩ ≡
```
aap, noot, mies: real;
maan zaag, fien, vier:integer;
```
◇
Macro referenced in scrap 1a.

Een macro kan op verschillende plaatsen in het document "gedefinieerd" worden. De inhoud van de verschillende definities wordt gewoon aan elkaar geplakt. De macro's worden automatisch genummerd (in het voorbeeld worden de nummers 1a t/m. 1e gebruikt). Het nummer is gebaseerd op het nummer van de bladzijde waarop de macro staat (bijvoorbeeld macro 6a staat op bladzijde 6).

1.2 AWK Om het logo te maken moeten er veel berekeningen gemaakt worden. Noch LaTeX noch Nuweb kunnen goed rekenen. Daarom gebruiken we een andere computertaal om de LaTeX code te genereren. Deze computertaal is AWK [1]. AWK, genoemd naar de eerste letters van de achternamen van de ontwikkelaars ervan (Aho, Weinberger en Kernighan) is een geïnterpreteerde, C-achtige taal, die vooral geschikt is om tekst strings te bewerken.

1.3 Vorm van het logo en ontwerp beslissingen Het logo bestaat uit de letters H, I en T, die samen een baan dragen die horizontaal begint en geleidelijk opstijgt. De baan bevat de horizontale balk van de H, de punt van de I en de horizontale balk van de T. Uiteindelijk ziet het er uit als in figuur 1. Het ontwerp wordt geparametriseerd, dat



Figuur 1. Het logo zoals dat met dit document gemaakt kan worden. Dit document maakt alleen de uitlijning van het logo. Het werkelijke logo is massief geel.

wil zeggen dat van de onderdelen zo min mogelijk de dimensies zelf aangegeven zullen worden, maar zoveel mogelijk de verhoudingen van de dimensies onderling met behulp van parameters. Op die manier kan, door het veranderen van een paar parameters, het karakter van het logo veranderd worden. Deze techniek is afgekeken van Knuth [2]. Het hier beschreven progamma maakt alleen maar een uitlijning van het logo. Om vanuit deze uitlijning tot figuur 1 te komen, heb ik nog veel werk met behulp van een grafisch programma moeten verrichten.

2 Maatvoering

2.1 Breedte, hoogte en indeling van de ruimte. Definieer de breedte en hoogte van het logo in willekeurige eenheden.

⟨definieer dimensies van het logo 5a⟩ ≡
```
breedte=1000
hoogte=500
```
◇

Macro defined by scraps 5abc, 6abcd.
Macro referenced in scrap 12a.

Bij het uitlijnen van tekst wordt de breedte van de letter `X` in het gebruikte font vaak als maat gebruikt. Ik doe dit analoog, maar gebruik de eerste letter van het logo, de `H`, als maat.

⟨definieer dimensies van het logo 5b⟩ ≡
```
hwijdte=0.15*breedte
letafst=1.2*hwijdte
```
◇

Macro defined by scraps 5abc, 6abcd.
Macro referenced in scrap 12a.

De letters en de baan hebben een niet-verwaarloosbare dikte. *hdikte* is de helft van de dikte.

⟨definieer dimensies van het logo 5c⟩ ≡
```
dikte=0.06*breedte
hdikte=0.5*dikte
```
◇

Macro defined by scraps 5abc, 6abcd.
Macro referenced in scrap 12a.

Zet desgewenst een box om het plaatje ter oriëntatie. Zoals we later zullen zien moeten we daarvoor AWK instrueren om een LaTeX regel af te drukken.

⟨put alles in het picture 5d⟩ ≡
```
# printf("\\put(0,0){\\framebox(%d,%d){~}}\n", breedte, hoogte)
```
◇

Macro defined by scraps 5d, 9bcd, 10ab, 11ab.
Macro referenced in scrap 12c.

2.2 Positionering van de elementen in het logo De letter `H` staat helemaal links, dan komt de `I` en dan de `T`. De afstanden worden door *hwijdte* en *letafst* bepaald. Let erop dat de letters dikte hebben.

⟨definieer dimensies van het logo 6a⟩ ≡
```
ipos=hdikte+hwijdte+letafst
tpos=ipos+letafst
```
◇

Macro defined by scraps 5abc, 6abcd.
Macro referenced in scrap 12a.

De baan begint links op een fractie *baanbhoogfrac* van de hoogte van het logo en eindigt in de rechter bovenhoek.

⟨definieer dimensies van het logo 6b⟩ ≡
```
baanbhoogfrac=0.60
baanbhoogte=baanbhoogfrac*hoogte
```
◇

Macro defined by scraps 5abc, 6abcd.
Macro referenced in scrap 12a.

2.3 Overige maatvoering Sommige elementen worden met bogen aan elkaar verbonden. In ieder geval wordt de `T` met de baan verbonden via een boog. De grootte van de boog wordt bepaald door de volgende parameter:

⟨definieer dimensies van het logo 6c⟩ ≡
```
boogparameter=0.04+breedte
```
◇

Macro defined by scraps 5abc, 6abcd.
Macro referenced in scrap 12a.

De hoogte van de `H` en de ruimte tussen de baan en de `i` moeten worden gedefinieerd.

⟨definieer dimensies van het logo 6d⟩ ≡
```
hhoogte=0.90*hoogte
ispleet=0.05*hoogte
```
◇

Macro defined by scraps 5abc, 6abcd.
Macro referenced in scrap 12a.

3 Maak de baan

De baan wordt beschreven door een combinatie van Bézier curven en tweedegraads polynomen. De bovenkant van de baan is een segment van een parabool met minimum op (0, baanhoogte + hdikte), die loopt door de rechterbovenhoek van het logo, het punt (breedte, hoogte). De onderkant van de baan heeft een minimum op (0, baanhoogte − hdikte) en loopt ook door de rechterbovenhoek van het logo. Om de berekeningen te vereenvoudigen gaan we over op andere coördinatenstelsels. Noem het oorspronkelijke coördinatenstelsel (x, y), dan gaan we voor de bovenkant van de baan over op (f, y_h) en voor de onderkant over op (f, y_l), waarbij:

$$f = \frac{x}{\text{breedte}} \tag{1}$$

$$y_l = \frac{y - (\text{baanbhoogte} - \text{hdikte})}{\text{hoogte} - (\text{baanbhoogte} - \text{hdikte})} \tag{2}$$

$$y_h = \frac{y - (\text{baanbhoogte} + \text{hdikte})}{\text{hoogte} - (\text{baanbhoogte} + \text{hdikte})} \tag{3}$$

$$\tag{4}$$

y_l en y_h zijn gelijk aan nul aan de linkerkant van het logo en 1 aan de rechterkant. Maak AWK functies om heen en weer te gaan van het ene naar het andere coördinatenstelsel.

⟨definieer AWK functies 7⟩ ≡
```
function f(x){
    return x/breedte
 }
 function x(f){
    return f*breedte
 }
 function yh(y){
    minim=baanbhoogte+hdikte
    return (y-minim)/(hoogte-minim)
 }
 function yl(y){
    minim=baanbhoogte-hdikte
    return (y-minim)/(hoogte-minim)
 }
 function yfromh(yo){
    minim=baanbhoogte+hdikte
    return yo*(hoogte-minim)+minim
 }
 function yfroml(yo){
    minim=baanbhoogte-hdikte
    return yo*(hoogte-minim)+minim
 }
```
◇

Macro defined by scraps 7, 8ab, 9a, 12b.
Macro referenced in scrap 12a.

In de alternatieve coördinatenstelsels wordt de baan beschreven als $y = x^2$. De afgeleide functie is dan $y' = 2x$. De raaklijn door de parabool in een punt (x_i, y_i) is dan $y = 2x_i x - x_i^2$. De raaklijnen door de parabool in de punten (x_1, y_1) en (x_2, y_2) snijden elkaar in (x_s, y_s), waarbij:

$$x_s = \frac{x_1^2 - x_2^2}{2\,(x_1 - x_2)} \tag{5}$$

$$y_s = -x_1^2 + \frac{x_1\left(x_1^2 - x_2^2\right)}{x_1 - x_2} \tag{6}$$

In de LaTeX picture mode kunnen we het paraboolsegment dus beschrijven als een Bézier curve door de drie punten (x_1, x_1^2), (x_s, y_s) en (x_2, y_2). We moeten wel nog de punten terugschalen naar het oorspronkelijke coördinatenstelsel.

De volgende AWK functie tekent een stuk van de bovenkant van de baan, tussen $x =$ `xb` en $x =$ `xe`. De getransformeerde coördinaten worden met een `t` aangegeven (bijvoorbeeld `xtb` is de getransformeerde van `xb`).

⟨definieer AWK functies 8a⟩ ≡
```
function hparaboolsegment(xb, xe){
  xtb=f(xb); xte=f(xe); ytb=xtb*xtb; yte=xte*xte;
  xtm=(xtb*xtb-xte*xte)/(2*(xtb-xte));
  ytm=0-xtb*xtb+xtb*(xtb*xtb-xte*xte)/(xtb-xte);
  xm=x(xtm); ym=yfromh(ytm)
  printf("\\qbezier(%d,%d)(%d,%d)(%d,%d)\n",
         xb, yfromh(ytb), xm, ym, xe, yfromh(yte))
 }
```
◇

Macro defined by scraps 7, 8ab, 9a, 12b.
Macro referenced in scrap 12a.

Nu komt een codefragment om een stuk van de onderkant van de baan te tekenen.

⟨definieer AWK functies 8b⟩ ≡
```
function lparaboolsegment(xb, xe){
  xtb=f(xb); xte=f(xe); ytb=xtb*xtb; yte=xte*xte;
  xtm=(xtb*xtb-xte*xte)/(2*(xtb-xte));
  ytm=0-xtb*xtb+xtb*(xtb*xtb-xte*xte)/(xtb-xte);
  xm=x(xtm); ym=yfroml(ytm)
  printf("\\qbezier(%d,%d)(%d,%d)(%d,%d)\n",
          xb, yfroml(ytb), xm, ym, xe, yfroml(yte))
 }
◇
```
Macro defined by scraps 7, 8ab, 9a, 12b.
Macro referenced in scrap 12a.

De bovenkant van de letter I moet gelijkvormig zijn aan de onderkant van de baan. Daarom definieer ik hier analoge functies voor deze letter.

⟨definieer AWK functies 9a⟩ ≡
```
function yfromi(yo){
  minim=baanbhoogte-hdikte-ispleet
  return yo*(hoogte-minim)+minim
 }
 function iparaboolsegment(xb, xe){
  xtb=f(xb); xte=f(xe); ytb=xtb*xtb; yte=xte*xte;
  xtm=(xtb*xtb-xte*xte)/(2*(xtb-xte));
  ytm=0-xtb*xtb+xtb*(xtb*xtb-xte*xte)/(xtb-xte);
  xm=x(xtm); ym=yfromi(ytm)
  printf("\\qbezier(%d,%d)(%d,%d)(%d,%d)\n",
         xb, yfromi(ytb), xm, ym, xe, yfromi(yte))
 }
◇
```
Macro defined by scraps 7, 8ab, 9a, 12b.
Macro referenced in scrap 12a.

3.1 Doe het! Probeer maar eens een baan te maken. Eerst tussen de poten van de H. De linkerpoot van de H neemt *dikte* in beslag en de rechterpoot begint bij hwijdte + 0.5dikte.

⟨put alles in het picture 9b⟩ ≡
```
hparaboolsegment(dikte, hwijdte);
 lparaboolsegment(dikte, hwijdte);
◇
```
Macro defined by scraps 5d, 9bcd, 10ab, 11ab.
Macro referenced in scrap 12c.

Nu de bovenkant van de baan tussen de H en de rechter bovenhoek:

⟨put alles in het picture 9c⟩ ≡
```
hparaboolsegment(dikte+hwijdte, breedte);
◇
```
Macro defined by scraps 5d, 9bcd, 10ab, 11ab.
Macro referenced in scrap 12c.

Nu de onderkant van de baan tussen de H en de rechter bovenhoek. Er moet een gat komen waar de verticale stok van de T in komt.

⟨put alles in het picture 9d⟩ ≡
```
lparaboolsegment(dikte+hwijdte, tpos-hdikte);
  lparaboolsegment(tpos+hdikte, breedte);
```
◇

Macro defined by scraps 5d, 9bcd, 10ab, 11ab.
Macro referenced in scrap 12c.

4 De letter H

Van de letter `H` is de dwarsbalk al in 3.1 neergezet. Nu de poten nog. Er is geen scheiding tussen de poten en de dwarsbalk. De linkerpoot:

⟨put alles in het picture 10a⟩ ≡
```
printf("\\put(0,0){\\line(0,1){%d}}\n", hhoogte);
  printf("\\put(%d,0){\\line(0,1){%d}}\n", dikte, yfroml(f(dikte)*f(dikte)));
  printf("\\put(%d,%d){\\line(0,1){%d}}\n", dikte, yfromh(f(dikte)*f(dikte)),
           hhoogte-yfromh(f(dikte)*f(dikte)));
  printf("\\put(0,0){\\line(1,0){%d}}\n", dikte);
  printf("\\put(0,%d){\\line(1,0){%d}}\n", hhoogte, dikte);
```
◇

Macro defined by scraps 5d, 9bcd, 10ab, 11ab.
Macro referenced in scrap 12c.

Nu de rechter poot:

⟨put alles in het picture 10b⟩ ≡
```
printf("\\put(%d,0){\\line(0,1){%d}}\n",
         hwijdte, yfroml(f(hwijdte)*f(hwijdte)));
  printf("\\put(%d,%d){\\line(0,1){%d}}\n",
         hwijdte, yfromh(f(hwijdte)*f(hwijdte)),
           hhoogte-yfromh(f(hwijdte)*f(hwijdte)));
  xcoor=hwijdte+dikte
printf("\\put(%d,0){\\line(0,1){%d}}\n",
       xcoor, yfroml(f(xcoor)*f(xcoor)));
  printf("\\put(%d,%d){\\line(0,1){%d}}\n",
         xcoor, yfromh(f(xcoor)*f(xcoor)),
         hhoogte-yfromh(f(xcoor)*f(xcoor)));
  printf("\\put(%d,0){\\line(1,0){%d}}\n", hwijdte, dikte);
  printf("\\put(%d,%d){\\line(1,0){%d}}\n",hwijdte, hhoogte, dikte);

```
◇

Macro defined by scraps 5d, 9bcd, 10ab, 11ab.
Macro referenced in scrap 12c.

5 De letter I

De letter `I` is een verticale “balk”, echter met een schuine bovenkant, die gelijkvormig is aan de onderkant van de baan. De functies *yfromi* en *iparaboolsegment* om de coördinaten van de bovenste hoeken van de `I` te berekenen resp. de bovenrand te tekenen zij al gedefinieerd in hoofdstuk 3.

⟨put alles in het picture 11a⟩ ≡
```
xli=ipos-hdikte; xre=ipos+hdikte;
printf("\\put(%d,0){\\line(0,1){%d}}\n", xli, yfromi(f(xli)*f(xli)));
printf("\\put(%d,0){\\line(0,1){%d}}\n", xre, yfromi(f(xre)*f(xre)));
printf("\\put(%d,0){\\line(1,0){%d}}\n", xli, dikte);
iparaboolsegment(xli, xre);
```
◇

Macro defined by scraps 5d, 9bcd, 10ab, 11ab.
Macro referenced in scrap 12c.

6 De letter T

De letter T is een verticale stok die een open verbinding met de baan heeft. De baan heeft al een "gat" waar de stok van de T tegenaan moet lopen.

⟨put alles in het picture 11b⟩ ≡
```
xli=tpos-hdikte; xre=tpos+hdikte;
printf("\\put(%d,0){\\line(0,1){%d}}\n", xli, yfroml(f(xli)*f(xli)));
printf("\\put(%d,0){\\line(0,1){%d}}\n", xre, yfroml(f(xre)*f(xre)));
printf("\\put(%d,0){\\line(1,0){%d}}\n", xli, dikte);

```
◇

Macro defined by scraps 5d, 9bcd, 10ab, 11ab.
Macro referenced in scrap 12c.

7 Implementatie

Het document dat u nu leest (`logo.w`) maakt een LaTeX bestand aan, `logow.tex` waarin het logo getekend wordt. Het document maakt ook een AWK bestand aan, `logow.awk`, waarmee de picTeX instructies gemaakt worden. De picTeX instructies worden met een `\input` instructie in `logow.tex` opgenomen. Met LaTeX en een printer driver kan het logo tenslotte op papier gezet worden.

Schrijf het LaTeX script. Het begint met een standaard preambule, waarin `unitlength` wordt ingesteld. Vervolgens wordt het te maken picture in een `center` omgeving opgenomen.

"logow.tex" 11c ≡
```
\documentclass[a5paper,landscape]{artikel3}
\pagestyle{empty}
\setlength{\unitlength}{0.1mm}
\begin{document}
\begin{center}
\mbox{\input{logopic.tex}}
\end{center}
\end{document}
```
◇

Schrijf het AWK script.

"logow.awk" 12a ≡
```
BEGIN{
  ⟨definieer dimensies van het logo 5a, … ⟩
  ⟨maak het picTeX bestand 12c⟩
  }
  ⟨definieer AWK functies 7, … ⟩
```
◇

Het AWK script moet vaak hele regels LaTeX tekst schrijven. Om dit te vereenvoudigen is er de volgende AWK functie:

⟨definieer AWK functies 12b⟩ ≡

```
function pl(regel){
  printf("%s\n", regel)
}
```

◇

Macro defined by scraps 7, 8ab, 9a, 12b.
Macro referenced in scrap 12a.

7.1 De structuur van het LaTeX bestand Een LaTeX bestand heeft een vaste structuur. Maak in de "body" van het document een gecentreerd "picture" met het logo.

⟨maak het picTeX bestand 12c⟩ ≡

⟨open het picture 12d⟩
⟨put alles in het picture 5d, … ⟩
⟨sluit het picture 12e⟩

◇

Macro referenced in scrap 12a.

Maak een gecentreerd picture van breedte *breedte* en hoogte *hoogte* eenheden.

⟨open het picture 12d⟩ ≡

```
printf("\\begin{picture}(%d,%d)\n", breedte, hoogte)
```

◇

Macro referenced in scrap 12c.

⟨sluit het picture 12e⟩ ≡

```
pl("\\end{picture}")
```

◇

Macro referenced in scrap 12c.

7.2 Zet alles in beweging Het voordeel van de Nuweb aanpak is, dat ook het script om alles in beweging te zetten in het document gevoegd kan worden. Hier komt een eenvoudige Makefile.

"Makefile" 13a ≡
⟨suffix regels 13b, … ⟩
⟨specifieke regels 14a, … ⟩
⟨regel om dit document af te drukken 14d⟩
⟨regel om het logo te bekijken 15b⟩
⟨regel om het document te previewen 15a⟩
◇

Suffix regels binnen een Makefile geven aan hoe bestanden met een bepaald achtervoegsel gemaakt moeten worden. Bijvoorbeeld: Als je een bestand met achtervoegsel `.dvi` nodig hebt, en je hebt een bestand met achtervoegsel `.tex`, dan moet je (in ons geval) LaTeX draaien met het `.tex` bestand als invoer. De suffix regel die dit recept opgeeft aan het make programma gaat als volgt:

⟨suffix regels 13b⟩ ≡
```
.tex.dvi:
        latex $*.tex
```
◇
Macro defined by scraps 13bc.
Macro referenced in scrap 13a.

Zo kunnen we ook suffix regels maken om een `.tex` bestand uit een `.w` bestand te maken (door Nuweb te draaien)

⟨suffix regels 13c⟩ ≡
```
.w.tex:
        nuweb $*.w
```
◇
Macro defined by scraps 13bc.
Macro referenced in scrap 13a.

Het LaTeX bestand met de picTeX code voor het logo, `logopic.tex` moet worden aangemaakt met behulp van het AWK script `logow.awk` dat op zijn beurt weer uit `logo.w` ontstaat.

⟨specifieke regels 14a⟩ ≡
```
logow.awk: logo.w
        nuweb logo

logopic.tex: logow.awk
        gawk -f logow.awk >logopic.tex
```
◇
Macro defined by scraps 14abc.
Macro referenced in scrap 13a.

Om het logo te kunnen afdrukken of met een previewer te bekijken, moeten we een `.dvi` bestand hebben.

⟨specifieke regels 14b⟩ ≡
```
logow.tex: logo.w
        nuweb logo

logow.dvi:logow.tex logopic.tex
        latex logow
```
◇
Macro defined by scraps 14abc.
Macro referenced in scrap 13a.

Maak de `.dvi` versie van dit `.w` document. LaTeX wordt verscheidene malen herhaald om er zeker van te zijn dat alle referenties goed zijn.

⟨specifieke regels 14c⟩ ≡

```
logo.dvi: logo.tex logopic.tex
        latex logo
        bibtex logo
        latex logo
        nuweb logo
        latex logo
```

◇

Macro defined by scraps 14abc.
Macro referenced in scrap 13a.

Druk het Nuweb document af. Hier gebeurt dit met dvips en het Unix programma `lpr`. Voor andere operating systems moet deze regel waarschijnlijk worden aangepast.

⟨regel om dit document af te drukken 14d⟩ ≡

```
printnu: logo.dvi
        dvips -f logo.dvi |lpr
```

◇

Macro referenced in scrap 13a.

⟨regel om het document te previewen 15a⟩ ≡

```
viewnu: logo.dvi
        xdvi logo.dvi
```

◇

Macro referenced in scrap 13a.

Bekijk het logo op het beeldscherm. Ook deze regel moet waarschijnlijk aangepast worden op andere computers.

⟨regel om het logo te bekijken 15b⟩ ≡

```
view: logow.dvi
        xdvi logow.dvi
```

◇

Macro referenced in scrap 13a.

8 Conclusies

Het beginsel van parametriseren zoals dat door Knuth is beschreven, wordt hier nog maar beperkt toegepast. De parameters in deze toepassing beschrijven alleen de verhoudingen tussen lengtes en breedtes. Je zou ook parameters kunnen verzinnen als *rondheid* waarmee je regelt of en hoe de verschillende onderdelen met rondingen aan elkaar veronden zijn inplaats van rechte hoeken.

De *literate programming* techniek is geschikt om complexe programmeertaken, waarbij meerdere programmeertalen door elkaar gebruikt worden, uit te voeren. Het "programma" bestaat uit één ordelijk document dat goed te raadplegen en aan anderen over te dragen is, terwijl toch de sterke eigenschappen van de verschillende programmeertalen gebruikt kunnen worden. In dit voorbeeld zijn de goede grafische eigenschappen van picTEX gecombineerd met de goede rekenvaardigheid van AWK.

Tenslotte laat dit document zien dat eigenwijsheid lonend is. Door geen Metapost te leren heb ik een eenvoudig probleem ingewikkeld kunnen maken en daar publiciteit mee gegenereerd.

Referenties

1. Alfred V. Aho, Brian W. Kernighan, and Peter J Weinberger. *The AWK Programming Language*. Addison-Wesley Publishing Company, 1988.
2. Donald Knuth. The concept of a meta-font. *Visible Language*, 16(1):3–27, 1982.
3. Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, May 1984.
4. Donald E. Knuth. *TEX: The Program*. Computers & Typesetting. Addison-Wesley, 1986.

9 Index

TEXniques

Zet tekst op een vaste plaats op een bladzijde

Paul Huygen

abstract

In dit artikel is beschreven hoe een LaTeX probleem werd opgelost door het aan de TeX kopstukken voor te leggen via de NTG mailing list. Het concrete LaTeX probleem was het plaatsen van tekst op een vaste plaats op een bladzijde.

Inleiding

Soms is het nodig om tekstmateriaal op een bepaalde vaste plaats op een bladzijde te krijgen (dus met vaste coördinaten ten opzichte van de randen van het papier). De aanleiding van dit artikel was, dat ik door een drukker papier had laten maken met mijn bedrijfslogo er op. Zuinig als ik ben, liet ik hem alleen het logo erop zetten en niet de tekst, zodat ik slechts voor één kleur hoefde te betalen (bijkomende redenen zijn, dat het mij gemakkelijk leek om achteraf te bepalen welke tekst ik er neer wil zetten, en dat de te gebruiken fonts overeenkomen met de andere fonts van het document). Nu zat ik dus met het probleem, hoe daar precies tekst onder te krijgen. Na enig priegelwerk lukte het mij een eigen *brief* stijl te maken waarin de tekst op de juist plaats werd gezet. Een oplossing die binnen een bepaalde LaTeX stijl de tekst op de goede plaats zet zal echter onder een andere stijl de tekst verkeerd neerzetten, omdat LaTeX altijd vanuit ingestelde marges werkt, en die marges zijn voor iedere stijl anders. Ook stijlopties kunnen roet in het eten gooien.

Vraag het de kopstukken

In de LaTeX literatuur die ik tot mijn beschikking had kon ik geen oplossing voor het probleem vinden. Een korte zoektocht door CTAN leverde ook niets op. Daarom stuurde ik een vraag aan de NTG discussielijst `TEX-NL` (op Internet locatie `http://www.ntg.nl/tex-nl.html` staat informatie over deze discussielijst en staat hoe je je er voor kunt opgeven). Ik stuurde de volgende vraag naar deze discussielijst:

[..] Het probleem is nu, dat ik alleen kan vinden hoe ik tekst kan positioneren ten opzichte van binnen de style file gedefinieerde objecten (bijvoorbeeld ten opzichte van de linker kantlijn en ten opzichte van `headsep` *en* `headheight`*). Als ik dan iets gemaakt heb dat binnen de brief stijl goed werkt, dan zal het dus niet binnen de article stijl werken. Is er een manier om bijvoorbeeld een mbox te plaatsen op een vaste positie ten opzichte van de linker bovenhoek van het papier?*

Oplossing1: Doe het zelf

Johannes L. Braams stuurde een antwoord waarin hij liet zien hoe je zoiets kunt programmeren.

[..] Alle 'standaard' document classes gebruiken dezelfde parameters om de linker kantlijn en de bovenkant van de tekst te bepalen. Voor de linker kantlijn moet je kijken naar: `\oddsidemargin` *(op oneven pagina's of alle pagina's als niet tweezijdig wordt afgedrukt)* `\evensidemargin` *(op even pagina's als tweezijdig wordt afgedrukt) Ook moet je rekening houden met een eventuele* `\hoffset` *(normaal gesproken nul, maar je weet nooit wat een package ermee uitspookt...)*
Voor de horizontale positionering moet je dus de volgende berekening uitvoeren:

```
\newdimen\hor@comp
\if@twoside
  \ifodd\c@page
    \hor@comp\oddsidemargin
  \else
    \hor@comp\evensidemargin
  \fi
\else
  \hor@comp\oddsidemargin
\fi
\advance\hor@comp\hoffset
```

Verticaal moet je rekening houden met:
`\voffset`
`\topmargin` *(De afstand tussen de bovenkant van de sprekende kopregel en de bovenkant van de bladzijde, verminderd met 1 inch)*
Als je de positionering doet vanuit de kopregel (dus aanpassen van de pagestyle) dan krijg je in ieder geval:

```
\newdimen\ver@comp
\ver@comp\topmargin
\ver@comp\voffset
```

```
\documentclass[11pt,a4paper]{artikel3}
%\documentclass[11pt]{article}
\makeatletter
 \newdimen\hor@comp
 \if@twoside
   \ifodd\c@page
     \hor@comp\oddsidemargin
   \else
     \hor@comp\evensidemargin
   \fi
 \else
   \hor@comp\oddsidemargin
 \fi
 \advance\hor@comp\hoffset

 \newdimen\ver@comp
 \ver@comp\topmargin
 \ver@comp\voffset
 \def\@oddhead{%
   \hskip-\hor@comp\raise\ver@comp\hbox
     {Stavast}\hfill\null}
\makeatother
\begin{document}
Op deze bladzijde staat \'e\'en ding vast.
\end{document}
```

Figuur 1. LaTeX bestand om de methode van Johannes Braams uit te proberen. Als het *documentclass* statement bovenaan het bestand verwisseld wordt met het weggecomentarieerde statement eronder, dan blijft het woord "Stavast" toch op dezelfde plaats staan.

In de definitie van je pagestyle zou je dan kunnen opnemen:

```
\def\@oddhead{%
  \hskip-\hor@comp\raise\ver@comp
    \hbox{je tekst}\hfill\null}
```

Ik heb dit niet getest dus het zal me niet verbazen als het hier of daar iets moet worden veranderd om het gewenste effect te bereiken. Maar het geeft volgens mij wel de richting aan waarin je moet denken.
Het kan zijn dat je nog een extra niveau van boxing nodig hebt of zo. [..]

Deze oplossing heb ik uitgeprobeerd met het LaTeX bestand van figuur 1. Het interessante van deze oplossing dat hij laat zien hoe je zelf zo een probleem kunt oplossen met behulp van TeX code.

Oplossing 2: Postscript gebruiken.

Siep Kroonenberg gaf een oplossing met Postscript. Voordat ik de vraag stelde was ik eigenlijk bang dat de oplossing in Postscript gezocht moet worden. Het nadeel daarvan vind ik, dat je dan aan Postscript fonts vastzit.

Siep Kroonenberg schreef:

Als je een base-35 Postscript font gebruikt zoals Times of Helvetica dan kan het heel simpel met een dvips header file:

```
%!
/bop-hook {gsave xxx yyy moveto
  /Times-Roman findfont 10 scalefont setfont
  (naam) show grestore} def
```

xxx en yyy zijn de gewenste coördinaten van de tekst, uitgedrukt in bigpoints (1/72 in). Voor PostScript is het nulpunt van het coördinatensysteem linksonder. Noem deze file bijvoorbeeld. naam.pro en geef dvips een parameter '-h naam.pro'.
Er is natuurlijk veel meer mogelijk als je handig genoeg bent met Postscript.

Als je met postscript fonts wil werken, dan is deze oplossing heel eenvoudig te realiseren.

Oplossing 3: Aanpassen van een bestaande package

Werenfried Spit had ooit een package gemaakt (`pcentre.sty`) om een figuur midden op een blad te krijgen. Hij dacht dat dit wel aan te passen was om tekst elementen op een willekeurige plaats op het papier te zetten, bijvoorbeeld door een *picture* te maken die even groot is als de bladzijde, daar in het tekst element op de goede plaats te zetten, en deze *picture* vervolgens met zijn package midden op de bladzijde te zetten.

Oplossing 4: Package textpos gebruiken

Piet van Oostrum en Erik Van Eynde wezen op het bestaan van package `textpos`. Dit macro blijkt precies te doen wat ik wilde. Het package is gemaakt door Norman Gray (`norman@astro.gla.ac.uk`). Het moet als volgt gebruikt worden:

- Haal van het CTAN archief het *textpos* pakket uit directory `macros/latex/contrib/supported/textpos/` en haal pakket *everyshi* uit `macros/latex/contrib/supported/ms/everyshi.dtx`.
- Installeer beide pakketten, zodat TeX ze kan vinden.
- Zet in de preambule van het document waarin het moet worden toegepast: `\includepackage[absolute]{textpos}`. De optie *absolute* zorgt er voor dat de tekst elementen inderdaad op een vaste positie op het papier komen, en niet op een vaste positie ten opzichte van de plaats waar de functie van *textpos* wordt aangeroepen. Er zijn nog twee andere opties, waarmee de layout gemakkelijker getest kan worden.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{hitpos}
\typeout{Package: 'hitpos'}
\RequirePackage[absolute]{textpos}
\newcommand{\plaatshit}{%
\setlength{\TPHorizModule}{1mm}
\setlength{\TPVertModule}{1mm}
\begin{textblock}{60}(9,34)
 \parbox{60mm}{ {\sffamily {\large
    Huygen Intelligente Technologie}
   \\ Achter de Kerken 35
   \\ 1391 {\small LC} Abcoude
   }
 }
\end{textblock}
}
```

Figuur 2. Package om een tekst onder een voorgedrukt logo te zetten.

- Leg (met het `\setlength` commando) de lengte vast van de lengte eenheden `\TPVertModule` en `\TPHorizModule`
- De tekst "Stavast" kan bijvoorbeeld op een vaste plaats op een bladzijde gezet worden met de volgende statements:

  ```
  \begin{textblock}{<hsize>}{<hpos>,<vpos>}
  Stavast
  \end{textblock}
  ```

  Hierbij zijn <*hpos*> en <*vpos*> de vaste coördinaten op de bladzijde van de linkerbovenhoek van een *box* waarin in ons geval de tekst "Stavast" komt te staan. <*hsize*> en <*hpos*> zijn uitgedrukt in `\TPHorizModule` eenheden en <*hpos*> is uitgedrukt in `\TPVertModule` eenheden.
- Er zijn opties waardoor met de `textblock` omgeving inplaats van de linkerbovenhoek een andere hoek of een andere plek kan worden vastgelegd.

Een voorbeeld

Om *textpos* voor mijn toepassing te kunnen gebruiken heb ik de stijl optie *hitpos* geschreven. De listing staat in figuur 2.

Deze stijl optie definieert het commando `\plaatshit` die de naam en het adres van mijn bedrijf op de plaats zet waarboven op het voorbedrukte papier het logo staat.

Conclusie

In de eerste plaats laat het beschreven voorbeeld zien hoe behulpzaam veel van de deelnemers aan de TEX-NL discussielijst zijn. Ik wil ze graag hierbij nogmaals bedanken. Verder laat het voorbeeld zien dat veel TEXnici oplossingen hebben ontwikkeld voor verschillende problemen en deze beschikbaar hebben gemaakt voor anderen. Tenslotte laat dit voorbeeld zien hoe er verschillende mogelijkheden zijn om een bepaald probleem op te lossen.

Achteraf moet ik bekennen dat ik de vraag eigenlijk niet had hoeven stellen als ik CTAN beter had bekeken. Er bestaat een zoekprogramma op het Internet, waarmee je met trefwoorden naar pakketten in CTAN kunt zoeken. Het staat op `http://www.ctan.org/find.html`. Als je daar als trefwoord *absolute* invult, dan krijg je een verwijzing naar het pakket *textpos*.

TEXniques

DTP with TEX

Roland Kwee
S.S.R. Kwee Computer Consultancy
Amsterdam
email: rolandkwee@acm.org

abstract

A set of simple macros is presented, in the style of modular programming, to make it easy to put texts at arbitrary positions on the page, without being restricted by formatting rules. This is not only useful for single-page documents like announcements and business cards, but also for designing stationery with letterheads or printing labels. The method is based on putting kerned texts in boxes with zero horizontal and vertical size and staying in vertical mode. This works with plain, and any other, TEX, and can be combined with any other document formatting.

keywords

programming, DTP, vertical mode

Introduction

TEX is designed to "write beautiful books" and is therefore usually run with a carefully designed set of formatting rules. The idea is that an author should be concerned with writing down content while leaving typography to the publisher. By separating content and typography it is easy to create large documents with a uniform and consistent appearance.

Using the predefined plainTEX or LATEX styles, it is easy to fill a document with content. Changing a style or format, or even creating a new one, is much harder: you need to do some TEX programming. This isn't anything trivial, as can be guessed from the typical TEX appearance of documents typeset with esp. LATEX.

Personally, I can live with the formatting that others made available. What I do want, however, is to personalize the appearance of my correspondence, i.e., I want to design my own letterhead. This involves more than just adding a page headline or footline.

Another problem is that formatting gets in the way of designing very simple documents consisting of a single page. For a flyer, a business card, address labels etc, line breaking rules and page breaking rules are irrelevant. Worse, all these formatting rules make it hard to put a text a little higher or lower, next to that other text, etc. In fact, in such cases we don't want the carefully designed typography for large documents. We want to explicitly break all the rules and to be able to put the texts at any arbitrary position on the page.

Traditionally, there is a big schism between desktop publishing (DTP) programs that allow you unlimited freedom for the placement of texts, and text processing programs like TEX that place your texts according to rigorous built-in typography restrictions. However, being a TEX proponent (bigot?), I also want to fill my DTP needs with TEX. This is not just to save the money for a DTP program license, but also because some of my documents, notably letters, need the capability of TEX for the body of the text, combined with DTP capabilities for the letterhead.

Programming obstacles

Over the years I have done a lot of TEX programming, varying from specifying a new page size, e.g., A4 instead of the default U.S. letter size of 8.5 by 11 inches, to writing macros, to making a "typography" to print a nice calendar from the output of a program written in C. Each time I had to fix lots of problems with horizontal and vertical mode, unwanted spaces, adjusting glue, and so on. I can program in many languages, but programming in TEX still has the most "magic". I must admit that I could solve practically all problems by careful reading of "The TEXbook". And it is clear that all those difficulties are somehow unavoidable when programming text with a program written with text. Some text is text, and other text is program, all in one file, even in one line or word.

Another, unrelated, problem is that it is still common practice to write TEX programs in "optimized" style, that is, optimized for the computer, not the human. Looking in a format or style file, or at the examples in "The TEXbook" makes me think of a Forth or Perl program, or a Postscript file, also called "write-only" files. The effect is that TEX programming is discouraged and restricted to the high class of TEXperts and TEXnicians.

Some programming that works with plain TEX does not work with LATEX, and vice versa. The reason is that LATEX does not only add high-level features like nice table commands and indexing, but also introduces new programming concepts like variables or lengths, different kinds of boxes, and all that crap. Lamport says in his LATEX reference manual *There is no easy way to tell whether a Plain TEX command will cause trouble, except by trying it.*

Keep it easy

According to Albert Einstein, we should *keep it as simple as possible* (though not simpler). He was much more clever than I am, so I stick with this. How can we make programming in TEX easy?

Here are my recommendations:

- *Program in plain TEX.* Knuth's book is an excellent reference for plain TEX programming. For LATEX programming you'd have to study the format file. Good Luck.

- *Stay in vertical mode.* A TEX document always starts in vertical mode. Basically, you leave vertical mode when you start a new paragraph, or by an `\indent` or `\noindent`. Hence, all macros defined before the first paragraph are in vertical mode. If you stay in vertical mode, you don't have to worry about getting an unwanted space, or having to end lines with a `% comment`. It is essential to avoid unwanted spaces, if you want to have precise control of the placement of a text on the page. An unwanted space means glue, need I say more?

- *A macro should only do one simple thing.* This should not need further explanation. For those who need some: this is to enable reuse of the macro instead of reinventing the wheel.

- *Combine simple macros to achieve complicated goals.* This is also called breaking down a complicated problem into several simpler problems. TEX programming is difficult enough, so even simple problems can be challenging. Personally, I would not even try to solve complicated problems directly with TEX programming.

- *Group related macros into a module.* As several simple macros are needed to solve a real world problem, such macros should be grouped and kept together as a module. All globally visible names should begin with the name of the module. All macros should be in a file with the name of the module, and nothing more should be in that file. This way, a document can safely `\input` a module without naming conflicts (assuming you name the modules suitably).

- *Add sufficient documentation.* As a minimum, a file should begin with a comment describing its purpose. Each macro should have a description of its function, parameters and usage. Each trick should have ample explanation.

- *Use indentation.* There is really no justification for having to guess the level of parenthesation in effect at a word in a macro.

Easy macros

The task with DTP is to put a text on a particular position on the page. The macro DTPVPOS does just this. Actually, there is only one macro of one line in this DTP package:

```
% file: dtp.tex
% purpose: DTP capability for TeX

% V P O S
%
% Returns a vbox with zero dimensions,
% containing a word or vbox #3
% at offset (x=#1, y=#2).
% See: The TeXBook, appendix D, p. 389.
%
\def\dtpvpos#1#2#3{
% Place word at (x,y). #1=xpos #2=ypos #3=word
  \vbox to0mm{\kern#2\hbox{\kern#1{#3}}\vss}%
  \nointerlineskip
}
```

It takes three parameters: a text, an X and a Y coordinate. The text can be a `\vbox`.

The macro returns a `\vbox` of size zero. It is assumed that the macro is called when TEX is still at the starting position at the top-left corner of the document. After the macro call, TEX has not moved a bit, so subsequent calls to this macro can be made.

In the returned box the input box is kerned horizontally and vertically from the top-left corner to the specified position.

The macro uses a Dirty Trick from "The TEXbook", so I don't need to explain here why `\nointerlineskip` is used. Therefore I did not apply the indentation recommendation here.

The macro could also have been written with an hbox, using `\rlap`. This is left as an exercise for the masochist, because it is much better to work in vertical mode as explained before.

The description of the macro isn't really complete without an example of its use. Here is the start of the design of a pamphlet to announce a concert.

```
% Concert announcement of
% the ensemble ''encore''.
% Status: unfinished.

\input dtp

\parindent=50mm
% indentation of first line of a paragraph

\font\fa=pctu8r at 30pt
```

```
% pctu8r=Cheltenham-Ultra

% Define the musical part of the announcement
\def\vhead{
  \vbox{
    \dtpvpos{0mm}{0mm}{\fa Encore}
    \dtpvpos{0mm}{10mm}{%
       o.l.v. Ren\'e de Grote}
    % Put more texts here
  }
}

% Put together the various parts of
% the announcements
% C programmers would call this
% the ''main'' macro.
\def\vconcert{
  \vbox{
    \dtpvpos{20mm}{20mm}\vhead
    %\dtpvpos{0mm}{200mm}\vsponsor
    %to be defined
  }
}

% Execute the macros
\vconcert

\bye
```

It shows that it is easy to put a text of any font and any size, an important DTP feature, at any place on the page.

When the pamphlet is finished, it would probably consist of a few top-level macros, like one for the name of the ensemble, the date and location, and another for the name and logo of the sponsors. Each of these macros would contain calls to DTPVPOS for the various text elements. A "main" macro would then call DTPVPOS to place the text blocks of each top-level macro to the proper places on the page. This way, a pamphlet can be designed hierarchically.

All the while normal TEX formatting can be applied either within a box that is passed to a DTPVPOS call, or after the last DTPVPOS call for non-DTP text. The latter method can be used for a letterhead.

Another "recursive" application is label printing. First, make a macro calling DTPVPOS several times to construct one label with its several pieces of text, like name, street, city, logo. Then, make a macro calling DTPVPOS several times, one for each label on the sheet, with the first macro as the text argument. If you have ever tried to make an `\output` routine for two columns of five labels on a sheet, you will find the DTP method much easier. It will work the first time. And it will be easy even if the labels are non-uniformly distributed over the sheet.

Conclusion

TEX programming can be easy, and still powerful, by applying commonsense programming techniques. The dirty tricks can be hidden in a macro. TEX can be used both for traditional consistent document formatting and for traditional non-consistent desktop-publishing-like page formatting.

It is hoped that this style of macro programming will be used in other TEX applications, so that I can understand those. I even hope that this will make the use of, and programming in, TEX more available to "the masses", and that we can use TEX also to create *masterpieces of the desktop publishing art!*

fonts

Het gebruik van KIX™ in TEX

Maarten Gelderman

abstract

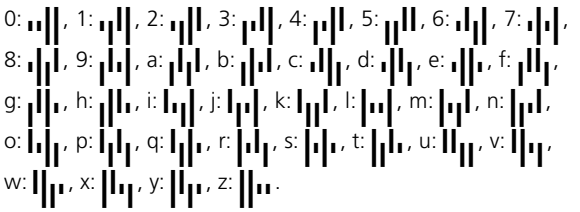
Recentelijk introduceerde de Nederlandse PTT de zogenaamde KIX™-code. Deze **K**lant**I**nde**x** is een streepjescode die gebruikte wordt ten behoeve van de automatische verwerking van poststukken. In dit artikel laat ik zien hoe KIX met relatief weinig moeite gebruikt kan worden binnen TEX en LATEX.

Inleiding

KIX is de nieuwe streepjescode van de PTT. Zoals bij de meeste streepjescodes ziet het eindresultaat er onleesbaar en best wel indrukwekkend uit, maar is het geheel eigenlijk niet meer dan een lettertype dat voor computers goed en voor mensen slecht leesbaar is. Een ander verschil met normale lettertypen is dat er slechts 36 tekens beschikbaar zijn: de 26 letters van het alphabet en de cijfers 0 tot en met 9. In principe komt het maken van de streepjescode slechts neer op het printen in dit lettertype. Om een beeld te krijgen van het te verwachten resultaat is in tabel 1 een printout van het lettertype opgenomen. Zoals daar te zien is bestaat iedere letter uit een viertal lange en korte strepen, die samen als een soort Morse-code het beoogde karakter weergegeven.

De eerste actie die we dienen te ondernemen om het font aan de praat te krijgen is het downloaden van het lettertype van de website van de PTT. Op deze website staat een specificatie van KIX, door de PTT ter beschikking gestelde hulpprogramma's en een bestand pcfonts.exe dat de lettertypes bevat. Dit laatste bestand dienen we te downloaden en op een Windows-machine uit te pakken. Van de bestanden die worden uitgepakt zijn er voor ons eigenlijk maar twee interessant: `kix-brg_.pfb` en `kix-brg_.afm`. Dit zijn respectievelijk het font in PostScript-format en de bijbehorende file met fontmetrics (in het eerste bestand staat de letter zelf, in het tweede bestand staat hoe groot de letters zijn).

Tabel 1. De karakters in het KIX-font



Het eerste wat we doen is de beide bestanden kopiëren naar een tijdelijke directory. Omdat de standaardnamen niet direct gemakkelijk leesbaar zijn, kopiëren we de bestanden naar respectievelijk kix1.pfb en kix1.afm.

Het lettertype beschikbaar maken

Helaas kan TEX niet direct met afm-bestanden uit de voeten. We moeten de adobe font metrics (afm) eerst omzetten naar TEX font metrics (tfm). Hiervoor gebruiken we het met de meeste TEX-installaties standaard meegeleverd programma afm2tfm en converteren het font zonder gebruik te maken van ook maar één enkele optie door simpelweg in te typen:

```
afm2tfm kix1.afm
```

De gegenereerde tfm-file plaatsen we in een geschikte directory (in mijn geval `/usr/TeX/texmf/fonts/tfm/kix`) en ook de pfb-file (die geen verdere bewerking hoeft te ondergaan) zetten we op de juiste plaats in de texmf-boom (bij mij `/usr/TeX/texmf/fonts/type1/kix`). Op vele systemen moet vervolgens de database met TEX-bestanden nog worden bijgewerkt, zodat TEX weet dat een nieuw lettertype beschikbaar is. Meestal geschiedt dit met het commando mktexlsr of via texconfig.[1]

Omdat de KIX-fonts postscript bestanden zijn, zullen we deze niet alleen voor TEX, maar ook voor de dviprocessor beschikbaar moeten maken. Ervan uitgaande dat we werken met dvips, doen we dit door aan het bestand psfonts.map de volgende regel toe te voegen:

```
kix1 KIX-Barcode-Regular <kix1.pfb
```

De macro's

Het lettertype zelf is nu gereed voor gebruik. We moeten alleen nog de noodzakelijke macro's aanmaken voor het genereren van de codes zelf. Ook dit is relatief eenvoudig. De hieronderstaande regels TEX-code volstaan en werken zowel in plain TEX als in LATEX:

1. Bij mij werkt het font overigens niet correct in combinatie met partial font-downloading, zodat indien in een implementatie partial downloading standaard aan staat de optie -jo aan dvips moet worden meegegeven.

```
1     % Wat lengtes uit de specificatie
2  \newdimen\kixsize
3  \kixsize=10pt      %readme-file bij de fonts
4  \newdimen\adresbreedte
5  \adresbreedte=10cm %paragraaf 5.1 specificatie
6  \newdimen\witruimte
7  \witruimte=3mm
8     %paragraaf 5 specificatie (min 2mm, max 15)
9
10    % Fontselectie
11 \font\kixfont kix1 at \kixsize
12
13    % Box met de nodige witruimte
14 \def\kixbox#1{\vbox{%
15    \vskip\witruimte
16        \hbox to \adresbreedte{%
17            \hskip \witruimte\kixfont #1\hfill}
18    \vskip\witruimte}}
19
20        % Het eigenlijke commando
21 \def\kix#1#2#3{%Zie paragraaf 2.1 specificatie
22    \kixbox{#1%De postcode zonder spatie
23        #2%huisnummer van maximaal vijf cijfers
24        X%Scheidingsteken,
25         %mag bij ontbreken van toevoegingen
26         %weg, maar dat moet niet
27        #3%Toevoegingen aan het huisnummer,
28          %maximaal 6 karakters
29        }}
```

De eerste zeven regels van dit bestand worden gebruikt om wat lengtes te specificeren. In de documentatie die met de fonts wordt meegeleverd staat dat het font in 10-punts grootte dient te worden gebruikt, verder staat in de KIX-specificatie dat we voor de code 10 centimeter ruimte moeten reserveren en dat om de code heen een witruimte van minimaal 2 en maximaal 15 milimeter moet staan.

Op regel 10 laden we het KIX-FONT op de vereiste grootte. We maken hierbij geen gebruik van PSNFSS, maar doen dit gewoon op de primitieve manier. Op deze manier werken onze commando's zowel in plain TEX als in LATEX. Bovendien lopen we niet het risico dat er vreemde lettertype-substituties optreden wanneer de KIX-code bij voorbeeld in een italic-omgeving of een omgeving met een gewijzigde lettertypegrootte wordt gebruikt. De streepjescode mag immers niet schuin gezet worden of worden vergroot of verkleind.

In regel 13 tot en met 17 definiëren we het commando `\kixbox`. Met dit commando zorgen we er voor dat er langs de rand van de streepjes code voldoende witruimte wordt geplaatst.

Tot slot maken we in regel 20 tot en met 26 het commando `\kix` aan. Met dit commando kunnen we de Nederlandse KIX-codes conform specificatie zetten. Het heeft drie parameters: de postcode (zonder spatie), het huisnummer en eventuele uitbreidingen op het huisnummer. Tussen het huisnummer en de uitbreidingen hierop plaatsen we conform de specificatie een X. Deze X is verplicht bij de aanwezigheid van een uitbreiding, maar mag ook bij afwezigheid worden gebruikt. We hoeven dus geen ingewikkelde toeren uit te halen. De Nederlandse codes zijn nu klaar voor gebruik. Om mijn eigen adres met code te zetten gebruik ik de volgende commando's:

```
Plantage Kerklaan 65/1\\
1018 CX Amsterdam\\
\kix{1018CX}{65}{1}
```

Met het volgende resultaat:
Plantage Kerklaan 65/1
1018 CX Amsterdam

Buitenlandse codes *mogen* van de PTT ook worden gebruikt. Zij bestaan uit een tweeletterige landcode, gevolgd door de postcode. Huisnummers en dergelijke worden bij deze werkwijze niet meer gebruikt. We kunnen hier natuurlijk ook een macro voor maken, of de code gewoon helemaal weglaten. Daar ik zelfs niet weet wat de tweeletterige code voor België is, heb ik maar voor de laatste oplossing gekozen.

Mogelijke uitbreidingen

Natuurlijk kunnen we het gebruik van KIX verder vergemakkelijken. Middels het ligatuurmechanisme van TEX zouden we huisnummer en uitbreiding hierop automatisch kunnen scheiden. We zouden TEX ook de juistheid van de postcode kunnen laten controleren en de lengte van de overige velden laten testen. Daar de gebruikte gegevens meestal toch uit een database zullen komen, lijkt dit echter zonde van de tijd. We hebben wel wat beters te doen: het splitsen van de straatnaam en het huisnummer in onze databases, want in dat opzicht valt deze standaard voor de gebruikers moeilijk gemakkelijk te noemen. Bovendien ga je je wel heel erg afvragen wat voor dubieuze statistieken de PTT over de door jou ontvangen post bij wenst te houden. Voor het sorteren lijkt de gevraagde informatie immers redelijk overbodig.

fonts

An Extended Maths Font Set For Processing MathML

Taco Hoekwater
Bittext VOF
Singel 191
3311 PD Dordrecht
The Netherlands
bittext@cybercomm.nl

abstract
Last years autumn, work started on a new set of mathematical fonts that are intended to cover the full range of characters included in MathML as well as those included in the proposals for mathematical extensions in the next version of Unicode.
This paper presents the first result of that work: A new Times-compatible maths font set consisting of about 1500 symbols and a few alphabets; along with a collection of TEX macros to use them.
These fonts are donated to the public domain by Kluwer Academic Publishers and are available in both MetaFont source and Adobe Type 1 formats.

keywords
mathematical typesetting, fonts, PostScript

Introduction

Typesetting mathematical material is and always has been a complicated matter. Not only does it require a rather specialized typesetting engine, but formulas also need some quite peculiar fonts. Even if the typesetting engine takes care of most of the complications involved with horizontal spacing, resizing of fonts and bouncy baselines, you still need fonts that contain the rather strange glyphs that are needed to display formulas beautifully.

Mathematical fonts have been a bit of a problem child in the past. Although there are quite a few font solutions around that can take care of relatively simple 'high-school' equations, there are a number of scientific fields that have had to improvise to convey the meaning of their notations simply because the needed glyphs were not available.

For other fields of science, the situation is marginally better: there are fonts available that contain the 'correct' characters, but only in a design that is visually incompatible with the normal text font of the article to be written.

The current work tries to cover all of the known fields that need special characters, so that there will be at least one *full* solution, compatible with Adobe's Times-Roman font.

Sources of information

Over the past few years, two groups have been working very hard to improve the situation, and their work gives valuable information regarding the needed glyphs.

The first group is the MathML working group, the group that is responsible for the coding of mathematics for the World Wide Web. This group has created a DTD fragment that allows both layout-based and content-based markup to be used for on-line mathematics [1]. The entity set used by this DTD fragment is essentially the same as an older ISO technical report, ISO 9573-13 [2].

The second group (STIX) is essentially a collaboration between scientists and publishers. This group [3] has tried to compile a comprehensive table of actually used/wanted mathematical glyphs, to be submitted to the Unicode organization [4, 5]. Their current table of glyphs lists about 2000 separate characters (including cyrillic, phonetics and chemistry).

The combined efforts of these two groups have resulted in about 1500 different glyphs for mathematical typesetting as well as the specification of about 20 alphabets that need at least all lowercase and uppercase latin characters.

A third interesting source of information are the specialized mathematical fonts that are already out there: fonts such as the StMary's Road Symbols and Waldi's Symbols, and all of the proprietary fonts that come with scientific software such as Mathematica and Scientific Workplace.

Getting started

In the autumn of 1998, Kluwer Academic Publishers in Dordrecht (The Netherlands) decided to adopt the MathML coding method for their new SGML-based production environment. Since the publisher currently still relies on paper for most of its income, a solution had to be found for the lacking characters & fonts. The design and implementation of a Times-compatible version of these fonts in Adobe Type 1 format has been commissioned to Bittext VOF, on the condition that the resulting fonts will be donated to the Public Domain.

The information sources that were to be used as a reference for the fonts to be designed were rather disorganized. The first job was trying to combine all possible information sources into one large collection of glyphs + descriptions that could then be used as a lookup–table, eliminating the need to ask questions to the two groups mentioned above for every second glyph.

Because of the extensive work already done by the STIX group, the part of merging the descriptions was fairly simple. But unfortunately the demonstration glyphs that were used by the STIX HTML table(s) were not really usable for the creation of high-quality fonts. The typical example glyph looked like the ones in Figure 1.

Figure 1 Original glyph description images

So, the next thing to be done was to find as many reasonably good renderings in already existing fonts as possible. This was done by generating bitmaps of all fonts that were known to me (using a few TEX macros and GhostScript). These bitmaps also show the boundingbox of the character as well as an 'em-square'. A few typical results of this method can be seen in Figure 2.

Figure 2 New glyph description images

Of course this method did not work for all possible glyphs, but at least it helped in deciding how to design the unknown glyphs (of which most were, of course, related to an already existing character).

After the bitmap-generation was done, the HTML lookup table was reorganized into blocks of glyphs that all had the same mathematical logical interpretation: one HTML file for arrow relations, one for binary relations, one for large operators, etc. This reorganization was necessary because one wants to design all similar glyphs at the same time to improve consistency. At that time, the original STIX table was organized according to the (tentative) Unicode positions, without any logical structure.

A piece of the new tables can be seen Figure 3. With all this work in place, it was possible to start the actual implementation.

	xme/179				
E23E		R	cudarrl		left, curved, down arrow
E23F		R	cudarrr		right, curved, down arrow
E240		R	harrcir		left and right arrow with a circle
E241	stmary/95	R	roarr	S\rightarrowtriangle	right open arrow
E242	stmary/94	R	loarr	S\leftarrowtriangle	left open arrow
E243	stmary/93	R	hoarr	S\leftrightarrowtria*	horizontal open arrow
E244	msam/32	R	zigrarr		right zig–zag arrow
E245		R	nvhArr *		not, vert, left and right double arrow
E246		R	nvrArr		not, vert, right double arrow
E247		R	nvlArr		not, vert, left double arrow
E248			angzarr		right angle with down zig–zag arrow
E249	qua6/126	R	curarrm		curved right arrow with minus
E24A	qua6/96	R	cularrp		curved left arrow with plus
E24B		R	ufisht		up fish tail
E24C		R	dfisht		down fish tail
E24D		R	rarrsim		right arrow, similar

Figure 3

Creating the fonts

A choice had to be made. The requested output format for the fonts was Adobe Type 1 [6], so two different workflows were possible:

1. Use an interactive editor such as Fontographer or FontLab to create the requested PFB files directly.
2. Use MetaFont and convert the result using Richard Kinch's MetaFog (see [7] and [8] for details of this conversion process and the programs involved).

Using an interactive editor has three big advantages:

- It generally works faster than programming in MetaFont;
- it is very simple to 'steal' shapes from existing fonts;
- and it is possible to edit all glyphs in one font file at the same time (MetaFont is limited to 256 characters per font).

The drawback of using an editor is the output, which is essentially write-only. It's not possible to re-use the created glyphs to create for example a sans–serif version.

MetaFont's advantages are

- its programmability: using dedicated macros in MetaFont it is easier to remain consistent.
- The MetaFont code is re-usable simply by changing the underlying macros and parameters.

MetaFont turned out to be the winner, not because this particular job really needs extensive programming facilities, but because it seems very likely that we will want to redo the maths font set for different font families in due time. This is a job which will be much easier using existing MetaFont code than it would scratch in an interactive program. Because in the latter case we would be obliged to restart from scratch again.

Planned for the future are at least a 'bold math' version of the Times compatible fonts, a sans–serif version to be used with e.g. Helvetica and Frutiger, and a Computer-Modern compatible implementation.

The actual implementation

In the actual implementation, there are conceptually only a few fonts: one very large font containing all of the 1500 special glyphs and a few separate fonts for the different alphabets.

The symbolic font

As said earlier, MetaFont cannot really handle fonts that have more than 256 characters in them. For this reason, the source has been split into sub-font chunks that are somewhere between 200 and 250 glyphs each (in implementation order):

- Arrow relations and other arrow symbols
- Ordinary symbols such as harpoons and angles
- Binary relations
- Negated binary relations
- Binary operators
- Delimiters and other extensibles
- Dingbats and various ordinary symbols
- Large operators
- Accents

These groups are now in various states of completion, from almost completely finished (the arrows) to rough pre-implementations (the large operators). Figure 4 gives an idea of what the resulting fonts look like (this is page two of the font table for the binary relations, ranging from character 81 to 174)

All of the sub-fonts share a common MetaFont base file, and almost all character definitions are defined in a very indirect manner. For example, here is the code for character 84: 'geqslantdot':

```
beginchar(geqslantdot_slot,42hu,27vu,6vu);
geqslantdot;
endchar;
```

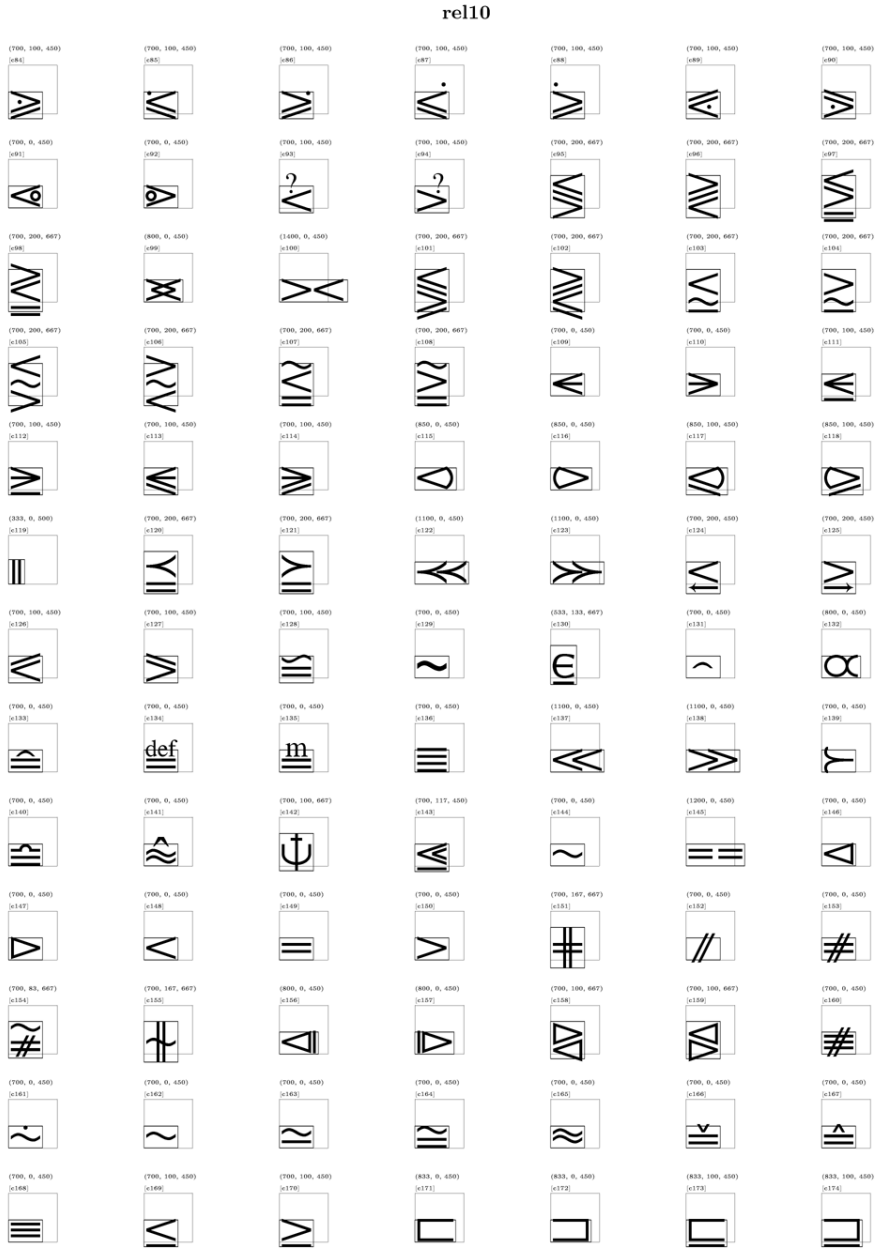


Figure 4

`geqslantdot_slot` is a number that gives the location of this character in the font. This list of numbers is another input file.

The macro `geqslantdot` is also defined in a separate macro file, and looks like this:

```
def geqslantdot =
leqslantdot;
```

```
hreflect;
enddef;
```

Of course, the referred–to `leqslantdot` represents the character that is the horizontal mirror of this one:

```
def leqslantdot =
leqslant;
dotat ((w-side-3thick,axis));
enddef;
```

And here, `leqslant` is defined in terms of `leq` and `eqslant`. All of the glyphs where this kind of indirection was feasible work this way: lots of glyphs are built up from (parts of) other glyphs, with added rotations or reflections around the axis.

The obvious advantage of this approach is that most of the sub-fonts consist of only a few dozen 'basic macros', who will be the only things that need to be changed for different versions to be implemented in the future. Of course, this scheme also helps out in the attempt to remain consistent in design throughout the sub-fonts.

Alphabets

From the discussions on the `math-font-discuss` mailing list, we came to a rather large list of alphabets that we seem to be needing:

- Math Italics
- Serif Text: Upright, Italics, Bold, Bold Italics
- Sans–serif Text: Upright, Slanted, Bold, Bold Slanted
- Greek Symbols: Upright, Italics, Bold, Bold Italics
- Blackboard: Upright, Slanted
- Fraktur: Upright
- Calligraphic: Upright, Bold
- Formal Script: Upright, Bold

Luckily, a lot of those alphabets can be borrowed or nearly borrowed from already existing (free) fonts. For instance, the Serif Text and Sans–serif Text can be taken from Times-Roman and Helvetica. Greek was borrowed from the Omega Unicode font, and Fraktur from the Euler fonts.

But others require quite some work. For instance, most current math texts use a Computer-Modern derivative for Blackboard Bold (`msbm` from the AMS fonts), which does not intermix well with Times. It is a bit crude and some of the capitals are somewhat different from Times; also all of the lower case letters are missing. A completely new version has been created, which results in:

This is 𝔑𝔈𝔚 𝔅𝔩𝔞𝔠𝔨𝔟𝔬𝔞𝔯𝔡 and this is 𝕆𝕃𝔻 𝔹<∋¬×∋⧹.

Likewise, new versions have to be done for the two required Script fonts as well as for the Calligraphic Bold version. These last three fonts are the final part of the work, and where not finished at the time of writing.

Current Status

There is still a lot of work that remains to be done in the TEX macro area: macros and virtual fonts have to be created before the fonts can actually be used in a sensible way from within TEX. Now that the encoding tables for the base fonts are reasonably fixed, this is rapidly becoming a first priority.

In MetaFont coding, about 90% of the work needed for the Times–compatible version is now complete. All of the symbols are available and the alphabets are well under way. The new maths fonts have therefore reached the point where user feedback is needed to finish the shapes and debug the metric information.

All current alpha and beta versions of the fonts (in both MF and PFB formats) and the related macros and metrics are available for public scrutiny from my web page:

`http://www.cybercomm.nl/~bittext/fonts.html`

In the current planning, there should be a user-level beta of the fonts as well as TEX macros available just before EuroTEX'99 in Heidelberg.

Literature

1. W3C REC-MathML-19980407, *Mathematical Markup Language (1.0) specification, W3C Recommendation 07-April-1998*.
   `http://www.w3.org/TR/REC-MathML/`
2. ISO/IEC TR 9573-13:1991(E), *Information technology – SGML support facilities – Techniques for using SGML – Part 13: Public entity sets for mathematics and science*.
3. `http://www.ams.org/STIX`
4. The Unicode consortium: *The Unicode Standard, Version 2.0* Addison Wesley, 1996 ISBN 0-201-48345-9.
   `http://www.unicode.org`
5. The Unicode consortium: *Unicode Technical Report # 8, The Unicode Standard, Version 2.1* Author Lisa Moore, September 4, 1998,
   `http://www.unicode.org/unicode/reports/tr8.html`. On-line publication.
6. Adobe Systems Inc: *Adobe Type 1 Font Format* Addison-Wesley, June 1995.
7. Richard J. Kinch: *Converting MetaFont Shapes to Outlines*. Paper presented at the 1995 TUG Conference in St. Petersburg, Florida, USA. Appeared in print in *Tugboat 16.3*.
8. Taco Hoekwater: *Generating Type 1 Fonts from MetaFont Sources*, 1998. *MAPS* #20, pp. 264–275.

Empty

Hans Hagen
PRAGMA ADE
Ridderstraat 27
8061GH Hasselt NL
pragma@wxs.nl

ConTEXt

TEX as presentation tool

an introduction to the ConTEXt presentation environments

abstract
In this article I will introduce a few styles I wrote on behalf of presentations. These styles are part of the ConTEXt distribution and can serve as an example of defining layouts in this macro package. More details can be found in the documented styles.

keywords
transparancies, presentation, pdf, ConTEXt

Introduction

Typesetting presentations is not the first thing that comes to mind when one reads books about TEX. Nevertheless I hope to convince the reader that TEX can produce wonderful presentations. In this article I will introduce six of the presentation environments that come with ConTEXt. Although each was written for a specific situation, they are general enough to be used in other situations as well. They also serve as an example of how to define styles in ConTEXt. These styles can best be used along with pdfTEX.

The original style

This style was used first at TUG–1997 in sunny San Francisco. Because a lot of information was presented, this style had quite a few navigational gimmicks. The look and feel was derived from the (interactive) module documentation styles.

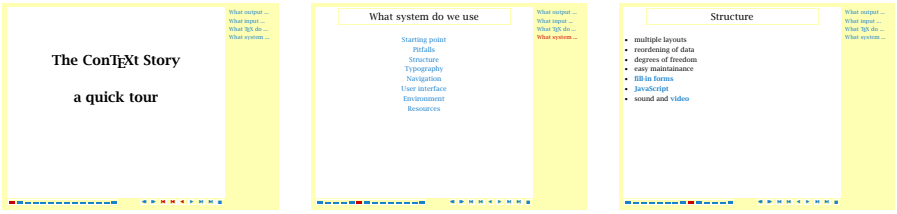


Figure 1 The original style.

This style is rendered in soft (sunny) yellow, blue, red and black. One might code such a presentation as follows:

```
\usemodule[pre-original]

\starttext
  \TitlePage {A simple presentation}
  \Topics    {Today's talk}
  \Topic     {First item}      .... some text ....
  \Topic     {Second item}     .... some text ....
  \Subject   {A subitem}       .... some text ....
  \Subject   {Another subitem} .... some text ....
  ....
\stoptext
```

Because chapters and sections don't make sense in presentations, we use more meaningfull sectioning commands. The `\Topic` commands generates a list of subjects (see second screen) and the topics themselves are visible in the margin. Long entries are automatically truncated. At the left bottom there is a status bar, at the right there are navigational buttons.

The green style

In 1998 I did several presentations on interactive documents and screen–document design. For that purpose I needed a simple style, which avoided the typical TEX look. Because I had METAPOST support at hand, I decided to use it for drawing the graphics (button shapes) that are part of the style.

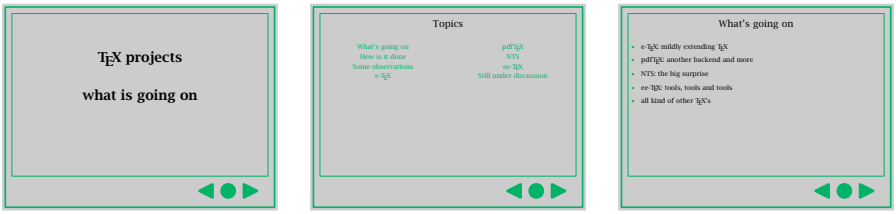


Figure 2 The green style.

In this style, lines, button shapes and bullets are dark green on a gray background. This style provides a lot of room for text. There is an optional extra button with the logical reference `Whatever`. One can use `\definereference` to assign a meaning to this reference if needed. I once used it to activate a movie and to launch programs.

The funny style

Because I was doing a few talks at TUG–1998 and was rather short on time, I decided to write an additional quick–and–dirty presentation style, which afterwards happened to be a rather nice one. Watch the darker segment within the dark red border: it moves with the incrementing page numbers. Of course this shape is generated by METAPOST.

Figure 3 The funny style.

In this style all non textual elements come in dark shades of red. The page counter is hardly visible in gray scales, but it is definitely there.

The colorful style

I wrote this style while preparing a tutorial for the UKTUG users group. Like the previous ones, it's screaming METAPOST again. I use primary colors only and the title page is a wink to the Dutch government which at the time I wrote this style was spending a huge amount of money on buying a Mondriaan painting (which was criticized in the press).

Like the green style, this one provides a lot of room for text. The list of topics is automatically typeset in columns when it becomes too large to comfortably fit in one column. The status bar at the bottom tells the audience how many screens there are left.

Figure 4 The colorful style.

The fuzzy style

The fuzzy style gets its name from the fact that each rectangle used in it is slightly different. I needed this style at a presentation for publishers but decided to use a similar style for the new interactive CONTEXT manual. Although TEX can do the job, I used METAPOST, which I happen to like more and more.

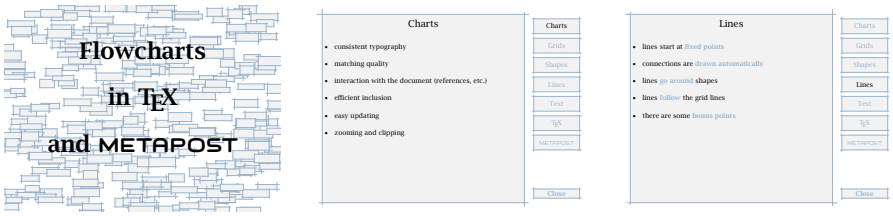


Figure 5 The fuzzy style.

This style is in light blue on light gray. Each presentation will look slightly different, due to the random lines. The list of topics is shown on the right. Make sure to use short topic titles and similar items. Hyperlinks and alike are shown in the same blue color.

The Polish style

One cannot go to Bachotek, the GUST paradise, and do a presentation without coming up with a POSTSCRIPT trick. This style was written in 1999 and uses the Antikwa Torunska, a font characterized by its prominent backward slant. In the background of the screen we see several O's, slightly distorted by METAPOST in a random way. This style should be used with itemized lists: they follow the background slant.

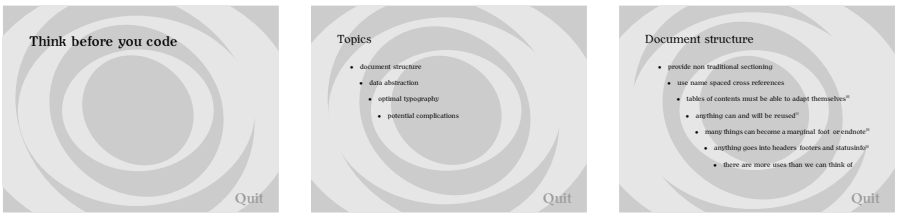


Figure 6 The Polish style.

This style has no buttons, apart from **Quit**. Clicking on the page moves you one page forward. This style is in grayscale.

More styles

Because I don't want to repeat myself too often, some more styles are under construction. These styles will be made public when I've used them a few times myself and when they are properly documented.

Commands

A presentation is set up with a minimum of commands. The title page is generated with:

\TitlePage *{title}*

One can use \\ to force a new line or start a subtitle. In case one wants a more complicated title page, there is \StartTitlePage ... \StopTitlePage.

In a presentation we often want to start with a summary. Such a summary is generated with:

\Topics*{title}*

Sometimes it makes sense to use the summary as title page. The summary lists the topics as defined by:

\Topic*{title}* ... text ...

In most cases, the text is an itemized list or a few text lines. The less text is used, the better it often suits the presentation. In principle one can use all CONTEXT commands here.

In a large presentation it makes sense to divide the topics into subjects using:

\Subject*{title}*

Depending on the style, the \Topic command generates a list of subjects.

Loading a style

Being part of the CONTEXT distribution these styles follow the module naming scheme, with filenames such as s-pre-01 and alike. For clarity we also provide more verbose names as shown table 1.

module name	file name
pre-original	s-pre-01
pre-green	s-pre-02
pre-funny	s-pre-03
pre-colorful	s-pre-04
pre-fuzzy	s-pre-05
pre-polish	s-pre-06

Table 1 The module names.

For instance, the fifth presentation style is loaded with:

\usemodule[pre-fuzzy]

This happens to mean the same as \usemodule[pre-05]; when loading a module we omit the leading s-.

Each module is documented and the typeset source is available as reference. The styles are not that complicated and the documentation takes more lines than the actual code. Although changing those styles is not that hard, users are strongly advised not to. It is far better to change layout characteristics in the document source.

Fonts

The styles use a 14.4pt body font. Some styles use the Lucida Bright fonts because they look very good on a computer screen. When these fonts are not available on the system, it makes sense to map them onto another font family. One way of doing this is adding the next line to the local cont-sys.tex file:

```
\definefilesynonym [font-lbr] [font-pos]
```

That way CONTEXT will use the three POSTSCRIPT fonts Times, Helvetica, and Courier, instead of Lucida Bright fonts. The three POSTSCRIPT fonts mentioned are nearly always available on the system.

Running METAPOST

Most of these styles rely on METAPOST. There are three ways to make sure that the graphics are generated:

- automatically run METAPOST at run–time
- let TEXEXEC take care of processing the graphics
- process the graphics manually between runs

The first alternative is the one I prefer. To enable this feature, one has to add `\runMPgraphicstrue` to the local `cont-sys.tex` file as well as to enable `\write18` in the (users) `texmf.cnf` file.

When TEXEXEC is properly set up, it will take care of generating the graphics, unless run time generation is enabled.

When both options are out of the question, the final alternative is to generate the graphics by hand. Just say: `mpost mpgraph` between runs, and the graphics will be there.

Graphics are stored in files with the name `mpgraph`. You can safely remove these files afterwards.

Page transitions

Fancy page transitions only make sense in presentations, and unfortunately the ones provided by the ACROBAT viewers are just ugly. Anyhow, one automatically gets them by saying:

```
\setuppagetransitions[random]
```

That way one gets random transitions. Resetting transitions is done by:

```
\setuppagetransitions[reset]
```

If needed one can specify transitions, but beware: these commands are very viewer dependant. Table 2 shows some alternatives. One can use numbers as well as sets. Using numbers is safer, i.e. less viewer dependant.

number				transition effects
1	2			{split,in,vertical} {split,in,horizontal}
3	4			{split,out,vertical} {split,out,horizontal}
5	6			{blinds,horizontal} {blinds,vertical}
7	8			{box,in} {box,out}
9	10	11	12	{wipe,east} {wipe,west} {wipe,north} {wipe,south}
13				{dissolve}
14	15			{glitter,east} {glitter,south}

Table 2 The transition effects.

The following settings are all valid:

```
\setuppagetransitions
\setuppagetransitions[1]
\setuppagetransitions[3,5,8,random]
```

Don't use transitions indiscriminately. Using transitions with for instance the Polish style spoils the design. I must admit that so far I never use transitions myself.

Slowly building up

The macro \presentationstep provides a basic slideshow functionality. It records portions of the page that will successively become visible. It can be used as follows:

```
\startitemize
\item eerste
\item tweede
\stopitemize

\presentationstep

\startformula
ax^2+bx+c
\stopformula

\presentationstep
```

When the document is opened, the two text fragments are hidden by a covering. Each page has its own stack of coverings. The reference NextStep can be used to hide the shield, for instance in:

```
\setupfootertexts[{\button{Show Up}[NextStep]}]
```

Special effects can be accomplished by setting up the related framed text:

```
\setupframedtexts
  [presentationshield]
  [background=color,backgroundcolor=red]
```

Expect this and similar features to turn up in future presentations. They are available when one loads the auxiliary module:

```
\usemodule[pre-general]
```

One can save some work by using the automatic covering mechanism. Just say:

```
\autopresentationsteptrue
```

For the moment this switch is not yet embedded in a more user friendly setup command.

More information

At www.pragma-ade.nl one can find the style documentation as well as the previous images in full color. The styles themselves are part of the standard CONTEXT distribution.

Hans Hagen
PRAGMA ADE
Ridderstraat 17
8061GH Hasselt NL
pragma@wxs.nl

ConTEXt
Typesetting Flow Charts
let TEX and MetaPost do the job

abstract
This article presents the ConTEXt module that deals with flowcharts and related types of charts. The charts are drawn at run–time by METAPOST in close cooperation with TEX. This not only gives us rather good graphics, but also provides a seamless integration of flow charts in documents, including hyperlink support and other fancy features.

keywords
Flowcharts, graphics, charts, METAPOST, ConTEXt

Introduction

This is just another story of TEX meeting METAPOST. This time we will focus on charts, especially flowcharts. In CONTEXT flowchart support is not part of the core functionality, but, at least for the moment, presented in a module. Therefore, before one can actually define a chart, this module must be loaded:

```
\usemodule[chart]
```

Once loaded, the user has access to the functionality described here. Before we go into detail on the features, we will say some words on history.

When dealing with graphics, it makes sense to use a drawing program. In fact, before we started using this module, we did use such programs, and they have without doubt their advantages. As soon as CONTEXT supported interactive documents, there were means to make graphics interactive, and as long as only a few graphics are involved, this mechanism works ok.

And then we suddenly had to make a document with thousands of pages and hundreds of often rather complicated flowcharts. Because these charts were tightly integrated in the main document, they not only had to be consistent in the use of fonts, but also had to be interactive and were to be presented both as a whole and in subchart parts. We wanted fonts, colors and the overall appearance as well as names of people, places, steps, activities and more to be consistent, especially because these charts are constantly updated.

I use the term flowchart here because I want to stress that this module typesets charts which cells are connected by lines. Our first application of this module concerned diagrams that expressed actions and relations between those actions, using some techniques originating years ago in programming environments: lines were not to cross, one should read from top to bottom and left to right, etc. However, the module presented here can be used to draw all kind of charts, and all kind of connections.

The grid

A flowchart consists of shapes, positioned on a grid, connected by lines. The grid enables the user to anchor the shapes and enables the drawing routines to determine connections. One can either explicitly specify the grid, or let it be calculated automatically.

Normally the grid is not visible, unless one enters test mode. The grid in figure 1 is the result of the definition:

```
\setupFLOWcharts
```

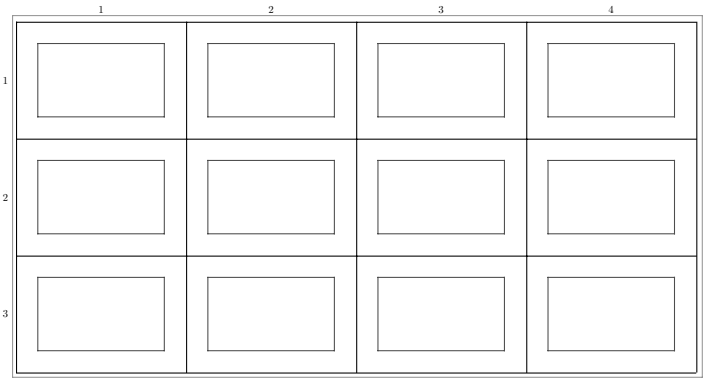


Figure 1 The grid.

```
[nx=4,
 ny=3,
 dx=2\bodyfontsize,
 dy=2\bodyfontsize,
 width=12\bodyfontsize,
 height=7\bodyfontsize,
 maxwidth=\textwidth]

\startFLOWchart [grid]
\stopFLOWchart
```

The most straightforward way of calling up this chart is by saying:

```
\FLOWchart[grid]
```

In figure 1 the inner rectangles represent the average size of the shapes. The lines around these shapes represent the grid cells. The outer rectangle shows the offset. Shapes are smaller than grid cells. This is necessary because connecting lines need some room. The offset is important, because when a connection follows the outer lines, a little extra space outside that line not only looks better, but also prevents the line from being clipped. It makes sense to keep the offset as well as the space between shapes constant across a document. The numbers are typeset outside the bounding box of the figure.

Grid cells are numbered from top to bottom starting at the left side, so the left topmost cell is (1, 1). Later we will see that because cells have names, these numbers play a minor role.

Shapes

A shape is something, typically a text, within a frame. The frame has certain dimensions and can have some color and background. In this respect it looks like the ConTEXt command `\framed`. The most important shapes have been assigned names as indicated in figure 2. There are more shapes, but they are identified by a number only. The total number of shapes will quite certainly increase. The shapes *up*, *down*, *left* and *right* are not really shapes, but lines that can be used to force a direction.

When no shape is specified, the default shape is used. One can change this default value with the `\setupFLOWshapes` command.

```
\startFLOWchart [cells]
  \startFLOWcell
    \name     {first}
    \location {1,1}
    \shape    {singledocument}
```

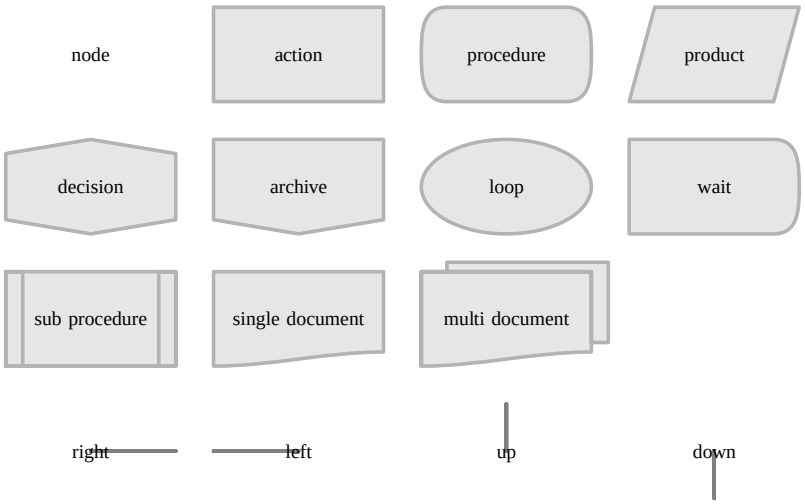


Figure 2 The shapes.

```
    \text     {not realy a document}
  \stopFLOWcell
\stopFLOWchart
```

A flow chart consists of cells. Each cell has a name, is positioned somewhere on the grid, has a certain shape, and normally this shape surrounds text. The shape is drawn by METAPOST, and the text is placed by TEX. Later we will see that there are some more fields to fill. Names are local to a chart.

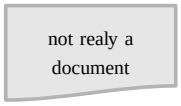


Figure 3

Connections

Shapes can be connected. As shown in figure 4 each shape has four connection points: top, bottom, left and right. When connecting shapes we refer to their logical names and specify two of the four directions.

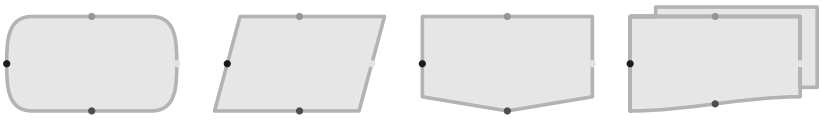


Figure 4 The connection points.

In figure 5 we see three connections. The lines have smooth curves and run across the grid lines. By using smooth curves, an option that can be turned off, the direction of touching curves is always clear. Here we use arrows. Smoothing, arrows and dashed lines are some of the attributes of lines.

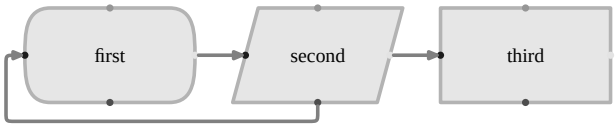


Figure 5 A few connections.

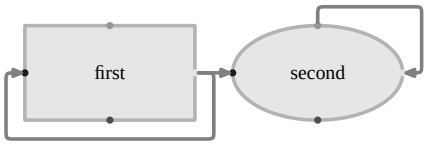


Figure 6 A few more connections.

There can be more than one connection per shape. When defining such a connection we first specify the direction. In this example `[rl]` means connect the right point to the left one, while `[tr]` results in a connection between the top and the right point. The second argument specifies the shape to connect to. As we can see, connections can point back to their origin shape.

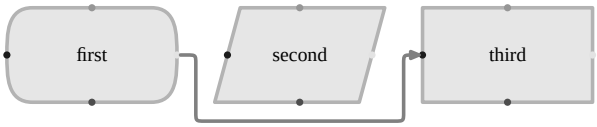


Figure 7 Going around shapes.

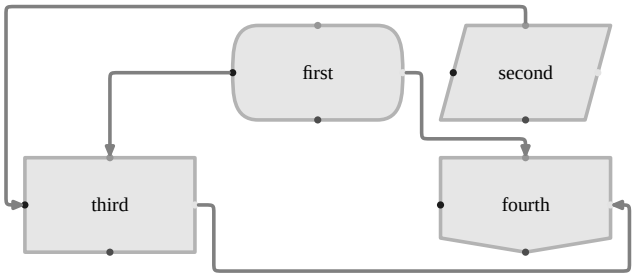


Figure 8 Following grid lines.

The connection drawing routines have a rather strong urge to follow grid lines. Figure 8 demonstrates this several times. From the first shape to the third one, we see that the connection takes the shortest route possible without crossing other shapes. I have to admit that the routines in themselves are rather stupid, but for normal use they suffice.

Generally speaking, when two lines end at the same point, it makes sense to connect these. When on the other hand lines originate at the same point or cross eachother, readers can get confused. Therefore such lines are drawn in such a way that they don't touch. In this respect, figure 9 demonstrates a less than optimal chart.

Adding text

In figure 10 we have added some comment to a connection. Like the dots at the connections, the point halfway the connection shows up in a special debugging mode. The comment will be placed relative to this point. In figure 10 this is to the left of the point.

It will be no surprise that a comment is defined using `\comment`. Comments can be anchored to eight locations, simply `l`, `r`, `t`, `b`, or a combination like `tr`.

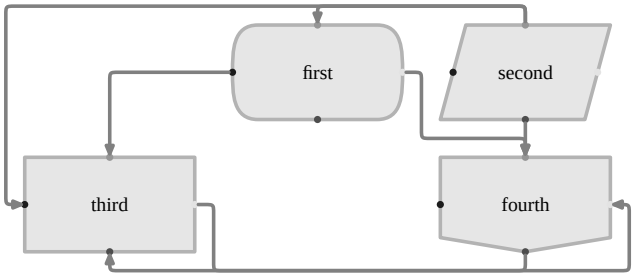


Figure 9 Confusing (crossing) grid lines.

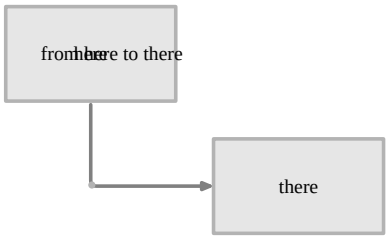


Figure 10 Comment to connections.

```
\startFLOWcell
  ...
  \comment [l] {from here to there}
  ...
\stopFLOWcell
```

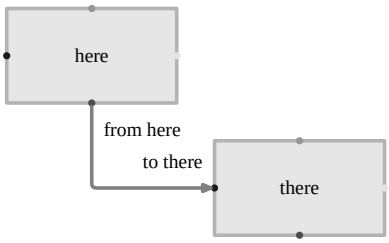


Figure 11 Labels to connection points.

We can also put labels at the connection points. Often this is preferred over comment halfway along a connection. Like comments, labels have a dedicated command. Here we specify the connection point `l`, `r`, `t` or `b`.

```
\startFLOWcell
  ...
  \label [l] {to there}
  ...
\stopFLOWcell
```

In figure 12 we see some text without any shapes around it. When shape `none` is specified, the whole shape area is available for text.

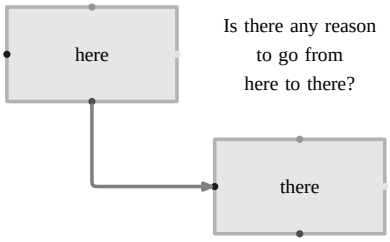


Figure 12 Text without shapes.

```
\startFLOWcell
  \shape    {none}
  \location {2,1}
  \text     {Is there any reason to go
             from here to there?}
\stopFLOWcell
```

One can force the alignment with the key characters `l`, `r`, `c`, `t` and `b`. So, the next definition only places text.

Inheritance

When explaining something by using a chart, we often show successive versions of the chart, where each version adds a new feature. To prevent us from retyping the same components again and again, it helps to partition the source code of the complete chart into subcharts. Inclusion of a part is straightforward: the subchart is called by name and positioned on the grid.

```
\startFLOWchart [include]

  \includeFLOWchart[labels][x=1,y=1]

  \startFLOWcell
    \shape    {none}
    \location {2,1}
    \text     {There is no reason to go
               from here to there!}
  \stopFLOWcell

\stopFLOWchart
```

The included sub chart has its own reference point, so one does not have to bother about positions.

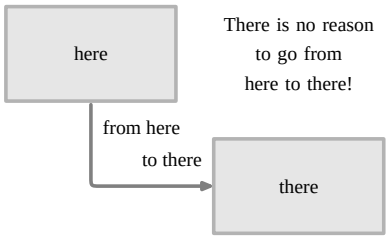


Figure 13 Sharing components.

How it works

The charting module, loaded by `\usemodule[chart]` is only responsible for the TEX part of the job, which means positioning text and graphics generated by METAPOST. The grid, shape and connection drawing routines are grouped together in a dedicated METAPOST module.

Because of the mix of TEX and METAPOST, and because we want to be able to scale charts, the buffer mechanism is used. The communication between TEX and METAPOST uses the METAPOST embedding macros that are native to CONTEXT. Additional communication from METAPOST to CONTEXT is handled in the module itself. This rather fuzzy description is visualized in figure 14. Depending on the general color settings, among the other processes involved are: color conversion and/or reduction to grayscales, and conversion to PDF. When watching this module in action, don't feel disturbed by the many steps involved.

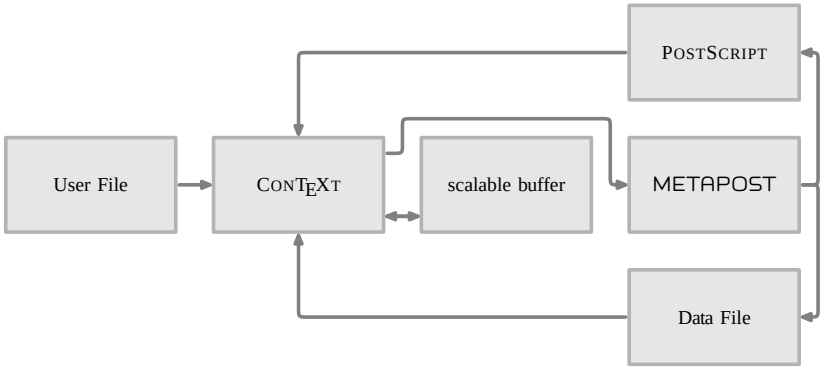


Figure 14 The process.

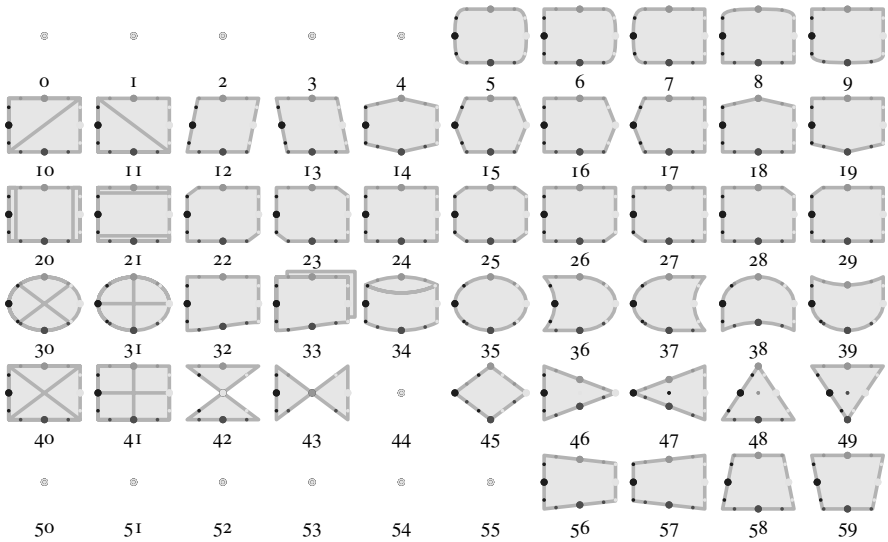


Figure 15 All shapes by number.

A few pages back we introduced the named shapes. There are however some more shapes. Each shape is identified by a number. In figure 15 all currently available shapes

are shown. The zero numbered shape is actually a shape, but with zero dimensions. It can be used as a meeting point for lines. The unused numbers can be used for new shapes. The maximum number of shapes is limited to 4095, which is a METAPOST limitation.

Bonus points

Sometimes charts can become rather large, due to the many shapes used or lines drawn. At the same time charts can become unclear because more than one connection starts or ends at a shape. Figure 16 shows a way out of this situation.

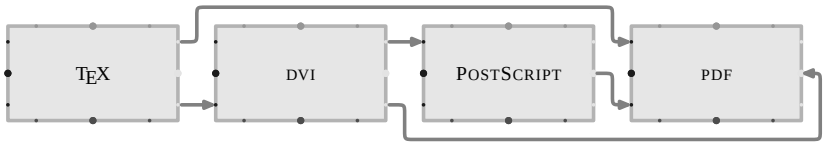


Figure 16 Even more points.

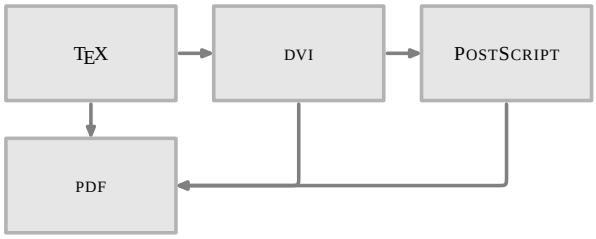


Figure 17 An alternative for figure 16.

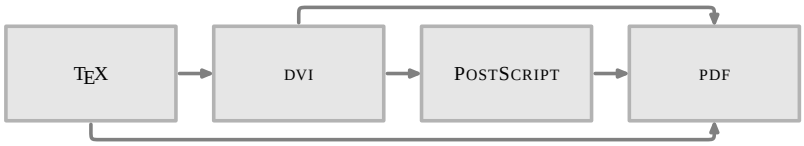


Figure 18 Yet another alternative for figure 16.

Defining such a chart is not so much harder than previous cases. Each left, right, top and bottom point has two companion points: positive and negative. In connections the left points are: `pl`, `l` and `nl`: positive left, left and negative left, so the first cell in figure 16 is defined as:

```
\startFLOWcell
  \name       {tex}
  \location   {1,1}
  \shape      {action}
  \text       {\TEX}
  \connection [prpl] {pdf}
  \connection [nrnl] {dvi}
\stopFLOWcell
```

Alternatively to `p` and `n` one may use `+` and `-`. As soon as the positive and negative points are used, the connection drawing routines will take into account the fact that they are off–center. This does not free users from thinking about better ways to draw such a chart.

Clip and focus

The flowcharter automatically calculates the size of the grid. When needed, one can force the dimensions and/or clip pieces of a chart. Figure 19 shows such a clip. This example also shows why the offset, the small area around the outer grid lines, is important. Figure 19 was produced while the next settings were in action.

```
\setupFLOWcharts
  [x=1,y=1,nx=2,ny=1]
```

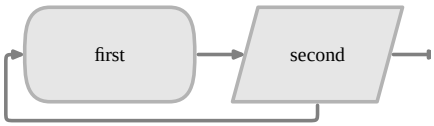


Figure 19 Clipping a piece of a chart.

Sometimes, for instance when explaining a chart, it makes sense to emphasize one or more particular cells. Therefore this module offers the ability to focus on cells. In figure 20 we see that the focus is on the cell with name `dvi`. This is accomplished by saying:

```
\setupFLOWfocus
  [framecolor=pragmacolor]
\setupFLOWcharts
  [focus=dvi]
```

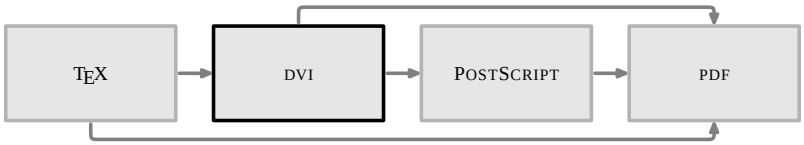


Figure 20 Gaining some focus.

Clipping and focus bring us to the third way of zooming in: autofocus. Figure 21 shows what happens when we say:

```
\setupFLOWfocus
  [framecolor=pragmacolor]
\setupFLOWcharts
  [focus=dvi,autofocus=dvi,nx=1,ny=1]
```

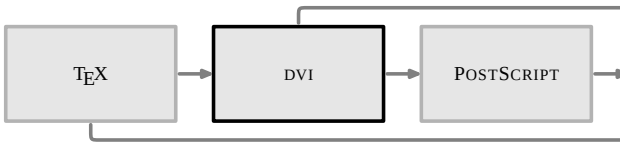


Figure 21 Applying autofocus.

In figure 21 we see both focus and auto focus in action.

Line types

As is to be expected, we can set up some characteristics of a chart, the shapes, the connecting lines, and the focus. When we want dashed lines and a different shape color, we just say:

```
\setupFLOWshapes
  [framecolor=pragmacolor]
\setupFLOWlines
  [framecolor=pragmacolor,dash=yes]
```

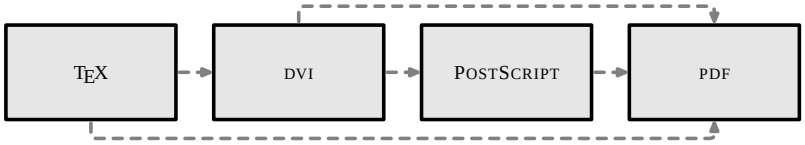


Figure 22 Dashed and colored lines.

We can change the characteristics at several levels: chart, line, shape and/or focus. In the near future some more options will be added. Once the METAPOST module is stable and documented, the graphic code will also be accessible. The formal definition of the four setup commands is as follows (these diagrams conform the CONTEXT conventions):

```
\setupFLOWcharts[..,..=..,..]

option            test
width             dimension
height            dimension
offset            dimension
dx                dimension
dy                dimension
nx                number
ny                number
x                 number
y                 number
autofocus         name
focus             name
backgroundcolor   name white
```

```
\setupFLOWfocus[..,..=..,..]

..=..    see \setupFLOWshapes
```

```
\setupFLOWlines[..,..=..,..]

corner          round rectangular
arrow           yes no
dash            yes no
radius          dimension
color           name FLOWlinecolor
rulethickness   dimension
offset          overlay none
```

```
\setupFLOWshapes[..,..=..,..]

framecolor          name FLOWframecolor
default             name action
background          color screen
backgroundcolor     name
backgroundscreen    number
rulethickness       dimension
offset              overlay none dimension
```

Overlays

Why should we limit ourselves to text? In ConTEXt most frames–related features can have overlays. Because in this flowchart module shapes are drawn by METAPOST, we use a slightly different approach: there can be overlays but they are sort of clipped by the shape. Figure 23 illustrates this.

Figure 23 Overlays.

There are two commands related to overlays. In our example we used:

```
\startFLOWcell
  ...
  \figure {cow}
  ...
\stopFLOWcell
```

which automatically scales figure *cow* to the shape size. An alternative command, that offers a bit more control, is:

```
\startFLOWcell
  ...
  \overlay {coward}
  ...
\stopFLOWcell
```

Here we call for an overlay, for instance defined by

```
\defineoverlay
  [coward]
  [{\externalfigure
      [cow]
      [width=\overlaywidth,
       height=\overlayheight]}]
```

The normal rules for overlays apply. The width and height of the overlay are available at the moment the overlay is typeset.

Interaction

One of the reasons for writing this module was that we wanted interactive flowcharts. The shape text can contain references.[1] The shape as a whole becomes a button as soon as we add the `\destination` entry:

```
\startFLOWcell
  ...
  \destination {destination}
  ...
\stopFLOWcell
```

1 Currently this only works ok when the chart is not scaled.

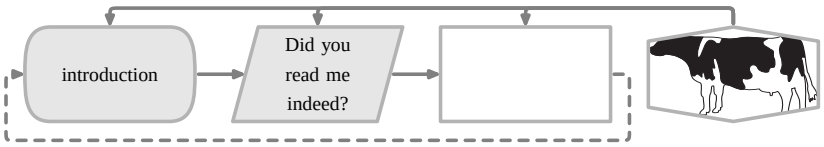


Figure 24 Overlays, help and buttons.

When read as PDF file, figure 24 shows quite a few interactive features. The first cell has a `\goto` included in the text, like:

```
\startFLOWcell
  ...
  \text {\goto{introduction}[introduction]}
  ...
\stopFLOWcell
```

The second shape is interactive as a whole. This is accomplished by saying:

```
\startFLOWcell
  ...
  \text        {Did you\\read me\\indeed?}
  \destination {introduction}
  ...
\stopFLOWcell
```

The third cell is a movie, which is only visible in suitable viewers. The movie is included as an overlay.

```
\defineoverlay
  [fun]
  [{\externalfigure
      [texwork.mov]
      [width=\framedwidth,
       height=\framedheight,
       preview=yes,
       repeat=yes]}]
```

The last cell is an illustration. Apart from the third cell, each cell has additional help information attached. Help information is defined with:

```
\startFLOWcell
  ...
  \help {identifier}
  ...
\stopFLOWcell
```

and defined by:

```
\starthelptext [identifier]
  ... text ...
\stophelptext
```

Helpinfo is placed by `\helpdata` and removed by executing the reference `HideHelp`. The macro returns a centered set of hidden help texts, that can be placed somewhere by the normal layout commands. The help information mechanism is independent from the flowchart module and keeps track of the pages where help is available.

The help information pops up when one points and clicks below a cell. By putting the help button there, it has less chance to interfere with other interactive things. Of course users should be made aware of this feature.

Summarizing, we can put text in a shape, provide one or more overlays to this shape, make cells and/or part of the text in a shape into a hyperlink, and show help when requested.

Splitting charts

Sometimes a chart does not fit comfortably on the page. In such cases, it makes sense to split up the chart. There is a dedicated setup command to serve splitting:

```
\setupFLOWsplit[..,..=..,..]

nx          number
ny          number
dx          number
dy          number
command     command
before      command
after       command
marking     on off
```

An example of splitting is:

```
\setupFLOWsplit
  [nx=5,ny=10,dx=1,dy=1,before=,after=\page]

\FLOWcharts[mybigflow]
```

For easy alignment of the split pages, cut marks are added. This can be turned off by setting `marking` to `off`. The `n` parameters determine the number of cells per split off section, and the `d` parameters specify the overlap: a value of 1 means that each split off section has one row and/or columns in common with its neighbour.

The splitter can be used with the split float placement macro:

```
\splitfloat
  {\placefigure{What a big flowchart this is!}}
  {\FLOWcharts[mybigflow]}
```

Here every flowchart gets an caption with a decent subnumber.

Other features

It is possible to predefine flow charts in a way similar to external figures. Currently this mechanism is under construction, so describing it would be a bit premature.

Crossing lines, which are often forbidden in charts, can be made less confusing by adding a gap in the lines to be crossed. This is one of the features that are already implemented but not yet accessible by the CONTEXT user interface.

Another feature, used in this document, concerns automatic down–scaling. As soon as we start scaling illustrations, we introduce inconsistent typography, especially in the bodyfontsize. Therefore, the flow chart macros, when told to, are able to automatically scale down in steps of 1 point.

At this moment we are using and testing this module in typesetting a quality assurance manual of about 3000 pages and with over 1000 (sub)charts. As soon as the TEX macros and METAPOST code are stable and tested, this module will be added to the CONTEXT distribution.

editors

ASCII editors for TEX on MS-Windows

Erik Frambach
Faculty of Economics
University of Groningen
E.H.M.Frambach@eco.rug.nl

abstract

There are many good ASCII editor programs available for TEX-users using MS-Windows. However, they all have there strong points and their weaknesses. In this article I will discuss a few of them.

What is ASCII anyway?

All TEX users know that TEX texts are written in ASCII. But what exactly do we mean by ASCII? It's not as obvious as you may think.

Strictly speaking, ASCII (which stands for *American Standard Code for Information Interchange*) is nut much more than the characters a–z, A–Z, 0–9, some punctuation marks, and a few other characters. Basically, everything that you can type in directly from a US keyboard. These characters are represented as codes 32 to 127, and a few more. This is quite sufficient for English text.

But what about accented letters and other characters that don't exist in English? They can get a slot somewhere in the 8-bit *extended ASCII* table that extends up to 255. So we have to define standards for those characters too. However, there are far more 'extended' characters than there are slots available in an 8-bit set. That's why *code pages* were defined. They describe the meaning of characters in a given 8-bit set. As you can imagine there are many of these, and naturally your ASCII editor program and your TEX compiler have to be (made) aware of what code page you are using. Here are some pitfalls that you have to be aware of:

- Code pages in MS-DOS, Windows, Unix and Apple are/can be different.
- Different versions of Windows (US, NL, PL, etc.) behave differently.
- Windows 95 differs from 98, which in turn differs from Windows NT.
- Using TEX *input encoding* or TEX code page conversion through *TCX* filters (a new Web2c feature) can make a big difference.

All in all it's easy if you stick to English, and it gets tricky if you use extended ASCII. Of course you could use TEX commands to specify 'special' characters, but something like `ge{\"\i}nd` becomes almost unreadable. In other cases such transliterations are simply impossible.

So how does this relate to editor programs? Good editor programs 'understand' code pages, e.g. when spell-checking or replacing strings. Recoding into different code pages is sometimes supported.

Another common problem in dealing with ASCII files is the way new lines are specified. On Unix a single CarriageReturn character is used. On MS-DOS and Windows a CarriageReturn followed by a LineFeed characters is used. On an Apple Macintosh a single LineFeed is used. Good editor programs can deal with all these formats. If not, the ASCII text will be nearly impossible to edit.

Criteria

When it comes to choosing an editor program, there are many criteria that may be important. Some are important to any user, others only to professional users. Some features are nice but not essential, some others may obstruct you from doing your job properly or efficiently. Below is a list of criteria that you may want to use when selecting a good editor program for your purposes:

- File size: if you want to be able to edit a huge (say, 10 MB or more) file, the editor has to be able to handle it, and still run sufficiently fast.
- Speed: some editor programs can do thousands of replacements in a fraction of a second, others will take a minute to complete the task.
- Syntax highlighting: if you are writing texts that contain keywords with special meanings (e.g. TEX commands) it can be very helpful if these are displayed in different colors.
- Column manipulation: some editors allow you to cut, paste or move columns of your text, while others can only move lines or paragraphs. If you are editing tables you may need this feature badly.
- Spell-checking: a built-in spell-checker may warn you if you are writing nonsense. However, this feature may slow the editor program down, or require lots of system resources.
- Word wrapping: if you type in text without looking at the screen it's very convenient if the editor program

automatically starts a new line when the current line is full. But you have to be able to switch this feature off in case you need to write long lines.

- Macro language: a good macro language (which is much more than just replaying key sequences) can help you automating tedious tasks.
- Conversion of CR/LF: if line endings can be selected, you will have less problems when you send the file to someone using a different operating system.
- Conversion between different code pages. If you get a file prepared on MS-DOS this can be an essential feature. If an IBM operating system (e.g. VM) using EBCDIC encoding was used to prepare a text you will also need to convert it to ASCII.
- Tab settings: tabs can be used to position texts. A good editor will allow you to set the size of tabs and (if required) it will show where tabs are inserted, and it can convert tabs to spaces.
- Auto save: it makes sense to save your text regularly (e.g. after 1000 key strokes or 10 minutes), so you don't loose everything in case the system crashes for some reason. It's nice if your editor program will do that for you.
- Multi-level undo: sometimes you may want to undo what you just did. Or even undo the last 10 or 100 modifications.
- Edit multiple files simultaneously: sometimes you may need to change something in a lot of files. It's very efficient if you can do that with only a few key strokes instead of file by file.
- Regular expressions: if the program supports these you have a very powerful tool for finding and replacing string in texts. Unfortunately the implementation of regular expressions differs significantly in different editor programs.
- Hexadecimal editing: 'real' programmers will need hexadecimal editing capabilities. Not really a TEX or ASCII issue, though, but very convenient for checking encodings.
- Dynamic Data Exchange: this feature can be used to set up inter-program communication. It can be a powerful tool in dedicated environments.
- Configurability: it helps a lot if you can configure the program just the way you like it.
- Available on multiple operating systems: if you use several systems it's very convenient to be able to use the same editor program on all systems.
- Price: some editor programs are free, others are expensive. However, an expensive program is not necessarily better than a cheap one.
- Trial version: before you buy something you want to see if the product really satisfies your needs.

If you care to think about it, I'm sure you can come up with another 20 criteria or more.

Some candidates

Below I've listed a few Windows (Win32 actually: they may not run on Windows 3.x) editor program that I think are worthy candidates as TEX text editor programs. This doesn't mean that these are the only ones you should consider, but I can assure you I've evaluated about twenty other editor programs that I discarded for various reasons.

E.g., if an editor program was not available from Internet, at the very least as a crippled shareware version, it was not even considered. If it failed to install properly, it was sent to the waste bin immediately. If it crashed within minutes, or if it kept popping up nag screens, it was thrown away at once.

The following editor programs passed the tests with good results and will be discussed in more details later:

- TSE
- PFE
- MED
- WinEdt
- UltraEdit
- Notetab

One more editor program should be mentioned here: Emacs. There are several varieties (NT-Emacs, XEmacs, to name but a few) of this extremely powerful editor program that is available on many platforms. However, because it is discussed in much more detail by Piet van Oostrum in another MAPS article, I will simply refer to that article.

On any Windows system there are always a few editor programs available:

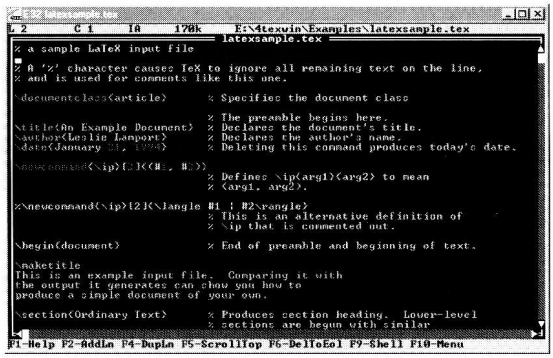
- Notepad
- Wordpad
- Edit

I will discuss the these editor programs as well, and compare them to the others, except for EDIT.COM. This program is remarkably poor: it doesn't even understand long files names. Therefore it doesn't qualify as a Windows editor program.

Pros and cons

In this section I will describe the editor programs and list their pros and cons. The descriptions are not an exhaustive checklists of all criteria, but rather a list of features that are very evident. If a certain feature is not listed, it doesn't mean it's missing, but it's less characteristic or not a very strong point.

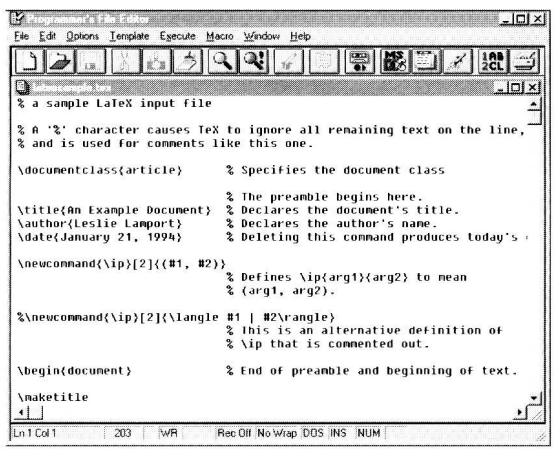
TSE (The Semware Editor)



+ very fast
+ good configurability
+ fine macro language
+ regular expressions (though not complete)
+ edit multiple files
+ German version available
± console application
− no DDE support
− commercial, trial version available
− syntax highlighting, but poor

The TSE editor program can be downloaded from `http://www.semware.com`.

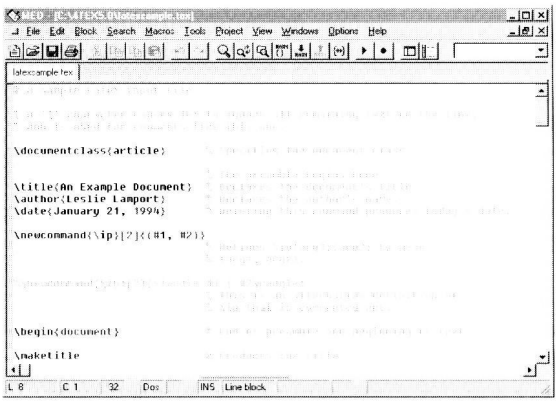
PFE (Programmer's File Editor)



+ highly configurable
+ completely free
+ supports DDE
− no syntax highlighting
− no regular expressions
− no macro language

The PFE editor program can be downloaded from `http://www.lancs.ac.uk/people/cpaap/pfe/`.

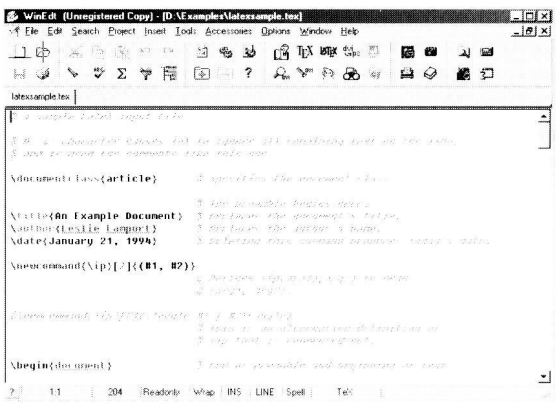
MED (Matthias' Editor)



+ reasonable syntax highlighting
+ support DDE for all commands
+ auto save
+ project manager
+ regular expressions
+ edit multiple files
+ can find matching [(< or) >] but not { }
+ can manipulate columns
+ supports templates
+ can run arbitrary tools (TEX compiler, viewer, BibTEX, etc.)
+ German version available
± cheap shareware
− no macro language

The MED editor program can be downloaded from `http://www.utopia-planitia.de`.

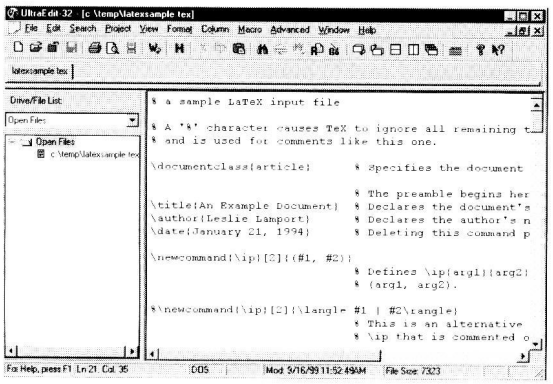
WinEdt



+ splendid syntax highlighting
+ many TEX features built in
+ can manipulate columns
+ understands character sets and TEX notation
+ built-in spell-checker
+ very powerful macro language
+ completely configurable, including menus
+ pre-configured for running MiKTEX
− shareware
− slow start-up, uses lots of system resources

The WinEdt editor program can be downloaded from `http://www.winedt.com/` or from any CTAN server (e.g. `ftp://ftp.ntg.nl`, directory `/tex-archive/systems/win32/winedt/`.

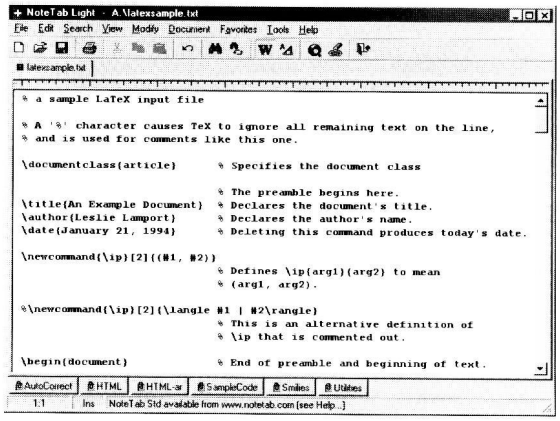
UltraEdit



+ can manipulate columns
+ built-in spell-checker
+ syntax highlighting
+ project manager built in
+ supports regular expressions
+ supports hexadecimal editing
+ supports templates
+ FTP client built in
+ supports file compare
+ many sorting options
+ splendid code page conversions
+ can run external programs and capture output
− shareware

The UltraEdit editor program can be downloaded from `http://www.ultraedit.com`.

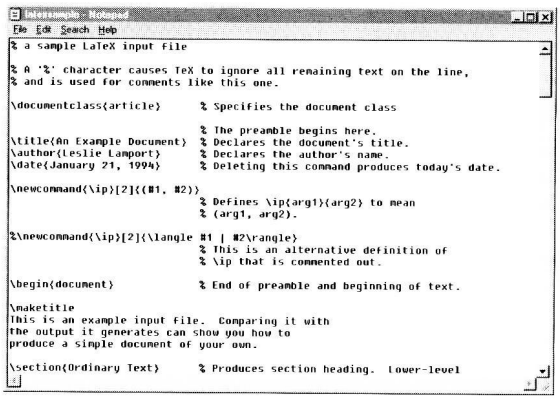
Notetab



+ free 'light' version, trial professional version
+ supports templates
+ good macro language (offers 'clipbooks')
+ nice macros for TEX/LATEX users available
+ supports regular expressions
+ can find matching [({ < or]) } >
+ good HTML editing features
+ supports several conversions
+ good configurability
− shareware

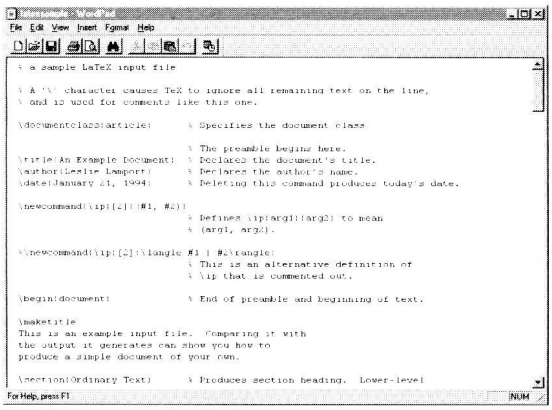
The NoteTab editor program can be downloaded from `http://www.notetab.com`.

Notepad



+ available for 'free' on every Windows system
− very few features
− older versions can't handle files larger than 32 KB
− can only edit one file at a time

Wordpad



+ available for 'free' on every Windows system
+ can handle ASCII, RTF and some MS-Word file formats
− can only edit one file at a time
± is it an editor program or a word processor? or neither? or both?

Conclusions

Looking back at this quick overview some conclusions can be drawn:

- All these editor programs have many features in common, but they are not equal. There are also many bigger and smaller differences.
- All these editor programs have there own strong points and weak points.
- There is no *ultimate* editor that does everything *and* is easy to use *and* is easy to learn *and* is cheap.
- There is no relation between price and performance. Cheap editor programs may well outperform expensive ones. But not necessarily.

Recommendations

So the question remains: what editor program should I choose? Here are a few recommendations to help you decide:

- Make a list of features that are important for your kind of work. Decide how essential these features are.
- Choose an editor that can do more than you currently need. You may want to use other features later.
- Invest time and energy in getting to know the program. You will find that you can do your work a lot more efficient if you know the short-cuts.
- Configure the editor program specifically for the tasks that you have in mind. Macros for often used functions can make your life a lot easier.

Because eventually the best editor program is the one that you are fully accustomed to!

editors

Using Emacs and AucTEX for preparing LATEX documents

Piet van Oostrum

abstract

Users of TEX and LATEX can be helped very much by an editor that knows about the specifics of these packages. For instance it can do syntax coloring so that the user sees immediately the difference between normal text and TEX commands; it can insert skeletons for often used commands and environments in order to prevent missing elements (e.g. missing \end parts), etc.

This article describes the use of the Emacs editing environment with the AucTEX package for the preparation of LATEX documents. The main characteristics of Emacs are discussed, followed by a more detailed description of the facilities that AucTEX offers to assist the author of LATEX documents. Finally we describe how AucTEX can be customized to support your own or external commands and LATEX packages.

keywords

Emacs, AucTEX, TEX, LATEX, packages, editing

What is Emacs?

Emacs (originally the name stood for "*Editing macros*") is basically an editor. But calling Emacs just an editor isn't fair: Emacs is much more. It can be used for reading and writing email and Usenet news, and as a Java development environment, to mention just a few applications. It would be better to call it as least an editing environment, or maybe a text work environment.

There exist several versions of Emacs, the most widely used one of which is GNU Emacs which was created by Richard Stallman. Even this one exists in two separate development streams, one maintained by Richard Stallman and the FSF (Free Software Foundation) and the other one (XEmacs) which started as an offspring of GNU Emacs at the now defunct Lucid Corporation. The main difference between these versions is the graphical user interface and the graphical possibilities which in XEmacs is more advanced. Most Emacs packages run on both versions, however. In the rest of the article we will use the generic term "Emacs" for both versions.

There are also several Emacs clones, which were primarily targeted at personal computers, because GNU Emacs and XEmacs are rather large. These clones are, however, not compatible with the main versions of Emacs, although some of them (e.g. Jed) have quite advanced possibilities, also for TEX and LATEX editing. We will not deal with these in this article, however. Both GNU Emacs and XEmacs have now complete versions running on The Win32 platform, and are therefore also available for the majority of PC users.

Emacs is a very flexible editing environment, which can be tailored to a lot of tasks. The main flexibility stems from the fact that Emacs is a *programmable* editor. It has a built-in Lisp interpreter with special primitives for text handling, buffer management, screen display, network functions (TCP/IP), operating system interface, and similar things. The basic operations and those where speed is important have been written in C, but all other functions are in Lisp. This makes it very easy to change the behaviour of Emacs, as the Lisp functions can be redefined at runtime, new functions can be added, and most of this happens automatically. Therefore Emacs is dynamically customizable and extendible.

Emacs also has an online documentation system, both for itself and for external programs. As an example, the documentation of LATEX commands is available and can be displayed while editing LATEX documents, even with defaulting automatically to the command or environment where the cursor is located.

Of course, there exist other editors which give similar support to the TEX or LATEX user, but most of these have been specifically written to give just this support and in most cases cannot be customized in the same flexible way as Emacs.

Emacs elements

Emacs uses the following terms:

Buffer A chunk of text, usually tied to a file. Editing happens in buffers, but not all buffers have to be files. Sometimes Emacs uses buffers just as a working space, and some buffers have their information from somewhere else, e.g. a directory listing.

Frame What most operating systems now call a *window*. When Emacs grew up it was used on plain old terminals

("glass TTYs") and the notion of windows as in GUI's was not known. And even Java calls them "Frame".

Window A part of a frame that displays a buffer. Windows in the Emacs sense cannot overlap, they tile the Frame (or the screen on a terminal). A buffer can be displayed in more than one window, even in different frames. The cursor position can be different in the different windows. In this way one can easily rearrange portions of text within the same file, or compare different parts of a file.

Major mode The way Emacs deals with a buffer. The major mode (or just mode) influences the way the text may be displayed (syntax coloring), what commands are available, which keystrokes are used for commands, which menus are available etc. There are many modes available, e.g. C-mode, Java-mode, Perl-mode, SGML-mode, plain-tex-mode, and latex-mode. The mode is usually inferred from the filename (extension), but can also be determined in other ways, for instance from the contents of the file. For the extension `.tex` the choice between plain-tex-mode and latex-mode will be made on the existence of `documentclass` or `documentstyle` in the file. It is also possible to state the mode explicitly in the file, or to change it manually after loading the file. Emacs comes with a standard mode for TEX and LATEX, but AucTEX is a more advanced one.

Minor mode A local change to the mode, to change some behaviour, e.g. `auto-fill-mode` for automatically wrapping words, `outline-mode` for showing outlines, or `abbrev-mode` for expansion of abbreviations. Minor modes can usually be toggled.

How Emacs interprets editing actions

Emacs uses lots of control characters, in conjunction with the Escape, ALT or Meta[1] keys for editing actions. The major editing actions are also available as menu entries. None of these have a fixed meaning, however. In line with the general customizability of Emacs every key and menu entry can be given any meaning. This is even true for character keys. In programming language modes (C, Java, Perl) the } key for instance not only inserts this character, but also positions it on a logical place on the line to support proper indentation. In AucTEX the quote character " generates the proper quotes to be used (“ or ”), in `auto-fill-mode` the space key triggers word wrapping etc.

Emacs uses *keymaps* to get this behaviour. A keymap is a table that binds keystrokes or combinations of them to Emacs functions. The functions can be written in Lisp or be built-in. Therefore one can let a key do almost anything. The "normal" character keys are bound to the function `self-insert-command` which just inserts the character.

AucTEX

AucTEX is a collection of Emacs lisp functions supporting a number of modes for editing TEX and LATEX documents. There are modes for plain TEX, LATEX, and texinfo (the Emacs documentation system), but others like a Context mode could easily be added. AucTEX has the following properties:

- Syntax coloring for macros, parameters of macros, comment, and environments. Some macros even influence the coloring, e.g., `\textbf{}` shows its parameter in bold.

- Menus and control keys for inserting chapter, section headers, etc., and environments. AucTEX knows the standard environments and can give the user a choice from these. When using a menu item to insert an environment (Insert Environment), one gets a menu with the known environment. When using the keystrokes to insert the environment one has to type in the environment name. At any point one can type the space key to get a list of possible completions of the name typed thus far. When only one completion is possible, AucTEX will give it, otherwise, it will complete as much as possible and present a list of the possibilities from which one can be chosen with the mouse or by pressing ENTER on the requires choice. But it is also possible to ignore these and type a completely different one, for instance an environment that you have defined yourself. The next time this will be amongst the choices presented (even in the menu). AucTEX knows more about the environments than just the name, but also what parameters are required, so it will prompt for them or reserve space for later insertion. Environments like `itemize` or `enumerate` will have an `\item` automatically inserted; as will happen for *description*, but in the latter case the `\item` will prompt for a label.

- Administration of `\label` and `\ref`.

- AucTEX knows about certain packages.

- Support of running of programs like LATEX, bibtex, makeindex, either on the whole document or parts of it. AucTEX makes intelligent guesses about which command is the next to use. For a new file or a file with recent changes this will be TEX or LATEX, after which bibtex may be necessary. Of course this is the default choice and can always be overridden.

1. One of the expansions of the EMACS acronym is "Escape Meta Alt Control Shift.

- Multi-file support. A file that is included in a main document can state which is its master file, such that running LaTeX will automatically give the main document as argument to the commands rather than the included document.

- Error processing. After running TeX or LaTeX, AucTeX parses the log file to find error message. With a keystroke or menu entry one can have AucTeX jump to the proper location in the source file.

Reference mechanisms

There are three mechanisms for easy navigation in a LaTeX file: Two come standard with Emacs or AucTeX; the third one is supported by an additional Emacs package.

The standard mechanism is supported by Emacs's imenu package, which gives a menu of the main divisions in a file. For programming languages, these are the functions and/or classes and methods, for LaTeX, this is the section structure. Each section-like command generates one entry in the menu, and selecting this menu entry causes the cursor to jump to that location in the file.

The second mechanism is the so-called *speedbar*: a narrow navigation frame similar to the left-hand panel in MS Window's Explorer. The speedbar frame can be started with a menu entry or an interactive command. When it starts up it displays the files in the current directory. Subdirectory entries can be opened to show their files or be closed, similar to Explorer in MS Windows. However, the same can be done for LaTeX files, or indeed any file type with imenu support. The opened document will then show the outline structure in the same way as the menu that is created by imenu. This gives an overview and makes it easy to navigate quickly through the document.

The speedbar entry for this article looks like the following (for some reason the entries are in reverse order):

```
0:[-] auctex.tex #
 1: > References}
 1: > Customizing Auc\TeX{}
 1: > Reference mechanisms}
 1: > Auc\TeX{}
 1: > How Emacs interprets editing actions}
 1: > Emacs elements}
 1: > What is Emacs?}
```

The third way is to use the additional Emacs package *RefTeX*. This package not only works with AucTeX, but also with the simpler standard TeX-mode that comes with Emacs.

RefTeX is a specialized package for support of labels, references, citations, and the index in LaTeX. RefTeX wraps itself round 4 LaTeX macros: `\label`, `\ref`, `\cite`, and `\index`. Using these macros usually requires looking up different parts of the document and searching through BibTeX database files. RefTeX automates these time-consuming tasks almost entirely. It also provides functions to display the structure of a document (the table of contents) and to move around in this structure quickly. The table of contents will be displayed in a separate window in the same frame as the LaTeX document, and you can jump from here to the location in the document with a single keystroke. It also works on multi-file documents, whereas imenu only supports a single file at a time.

This is the TOC of the current document:

```
1 What is Emacs?
2 Emacs elements
3 How Emacs interprets editing actions
4 Auc\TeX{}
5 Reference mechanisms
6 Customizing Auc\TeX{}
7 References
```

Customizing AucTeX

When starting to edit on a document AucTeX get its information from a number of Emacs Lisp files (`.el` files). Some of its intelligence is built-in, but this applies only to the general structure of TeX and LaTeX documents. For instance the fact that `\begin` and `\end` delimit environments; or the fact that a LaTeX document is characterized by a `\documentclass` or `\documentstyle` command.

For plain TeX documents, this is about all the knowledge that is available, but for LaTeX documents, AucTeX also reads the code in the Emacs lisp file `latex.el` and the ⟨classname⟩`.el` file, e.g. `book.el`. In the same way, `.el` files are read for documentstyle options, packages loaded with `\usepackage`, and included files. Some information can be automatically extracted by AucTeX, such as which commands and environments are defined in the files; other information must be provided by the package writer, such as special treatment of commands or environments. Therefore, AucTeX has two directories for these files, `auto` for automatically generated files and `style` for additional files.

Here is an excerpt of the file `latex.el`:

```
(TeX-add-symbols
  '("arabic" TeX-arg-counter)
  '("label" TeX-arg-define-label)
  '("ref" TeX-arg-ref)
  '("newcommand" TeX-arg-define-macro
          [ "Number of arguments" ] t)
```

It adds support for all the know LaTeX command and knows that `\arabic` should have a counter as argument,

`\label` defines a label, and `\newcommand` should prompt for "Number of arguments". The `label` support function remembers the label given, so that the `ref` function can supply a list of known labels.

Here is the main part of `book.el`. It sets the highest section-like structure to "chapter".

```
(TeX-add-style-hook "book"
 (function (lambda ()
  (setq LaTeX-largest-level
    (LaTeX-section-level "chapter")))))
```

We can also use these mechanisms to support our own packages. We give three examples: one for a simple command and two for an environment.

Suppose we have a package `qa.sty` which defines a command `\qa` with two parameters: a question and an answer. It will have a definition like

```
%% va.sty
\newcommand{\qa}[2]{....}
```

We now create a `va.el` file that defines the `\qa` command to prompt for the two parameters. These parameters are added literally (in braces).

```
(TeX-add-style-hook "qa"
  #'(lambda ()
    (TeX-add-symbols
     '("qa"
         TeX-arg-literal "Question"
         TeX-arg-literal "Answer"))))
```

There are other functions to add the arguments without braces, to ask for counters or other macros, etc. The file `qa.el` will be loaded automatically if `\usepackage{qa}` is present when it is in the proper directory (`style`).

The second example is similar, but for an environment rather than a command. This example is from the AucTEX manual. We want to have a package `loop` that defines an environment `loop` with three parameters: "From", "To", and "Step" are the prompts for the parameters.

```
\newenvironment{loop}[3]{...}{...}
```

We create `loop.el` as follows:

```
(TeX-add-style-hook "loop"
 (function
  (lambda ()
    (LaTeX-add-environments
     '("loop" "From" "To" "Step")))))
```

The last example is the modification of an existing environment. The package `enumerate` redefines the `enumerate` environment to accept an optional argument which describes the way the counter is to be formatted. We want the enumerate environment, when added by AucTEX to prompt for this parameter and insert it if a non-empty value is given. In this case we want to do some more non-trivial formatting, e.g. adding `\item` on the next line with proper indentation. We define our own function `LaTeX-enumerate-hook` to do the work. It reads the argument with the supplied prompt, test if an empty string is given, and if not, adds it to the document with square brackets added. We then insert the `\item` command with the function `LaTeX-insert-item`. As AucTEX starts a new line before the `\item`, we have to go back to the end of the previous line, delete the newline and any spaces inserted before calling this function.

```
; enumerate.el
(TeX-add-style-hook "enumerate"
 (function
  (lambda ()
    (LaTeX-add-environments
     '("enumerate" LaTeX-enumerate-hook
                "(Optional) Argument: "))))
  (defun LaTeX-enumerate-hook (name prompt)
   "Insert enumeration with optional argument."
   (let ((arg (read-string prompt)))
     (LaTeX-insert-environment name
             (if (string-equal arg "")
                nil
                (concat "[" arg "]" ))))
   (end-of-line 0)
   (delete-char 1)
   (delete-horizontal-space)
   (LaTeX-insert-item))
```

References

GNU emacs: GNU Emacs: `http://www.gnu.org/software/emacs/emacs.html`
XEmacs: `http://www.xemacs.org/`
AucTEX: `http://www.sunsite.auc.dk/auctex/`
Reftex: `http://www.strw.leidenuniv.nl/~dominik/Tools`

background: graphics

Tools for PostScript and pdf

Siep Kroonenberg
Kluwer, Dordrecht
siepo@cybercomm.nl

abstract

This paper explains why PostScript is interesting for TEX users, and describes various tools for working with PostScript, with special attention to Ghostscript. The paper concludes with a section on pdf, the derivative of PostScript which is destined to take over much of the role of PostScript in the prepress workflow.

keywords

PostScript, DSC comments, pdf, Ghostscript, graphics, conversions, eps inclusion, bounding box

At every turn, TeX users run into PostScript, the linchpin of the graphics industry. Imagesetters and high-end desktop printers understand PostScript; professional illustration software is geared towards generating PostScript. On Unix platforms PostScript tends to be the only adequately supported printer language.

The PostScript imaging model

PostScript is both a graphics format and a printer language. Its primary porpose is to describe objects on a page. A low-level printer might only understand descriptions in terms of printer dots; PostScript understands all of the following:

- a variety of color models
- paths, which may consist of curved and straight segments; the curved segments are described by either arcs or Bézier curves.
- several font formats; in the first place its own scalable Type1 font format, in which characters are defined by outlines
- bitmaps of arbitrary resolution in various color models; the PostScript interpreter is capable of resampling and rasterizing the bitmap to something matching the printer hardware
- clipping paths: objects will be visible only inasfar as they lie within the current clipping path.
- translating, scaling and rotating (affine transformations): it is possible to change the coordinate system on the fly. These transformations can also be applied to fonts.

PostScript-based illustration software

All these high-level features make it possible for graphics programs to adopt (a subset of) PostScript suitable as their native file format. Adobe Illustrator is the best-known of such programs, but all professional illustration programs (CorelDRAW, Macromedia FreeHand, Micrographix Designer and -Draw) are based on something close to the PostScript imaging model, and support Illustrator format almost as thoroughly as their own native format. Also worth mentioning in this context is Mayura Draw [11], a small inexpensive shareware PostScript-based Windows draw program.

Although traditionally Unix programs tended to support splines rather than Bézier curves, various draw programs for Linux/Unix (Sketch [12]; see figure 2, Killustrator [13], GYVE [14]) are now under development which do support Bézier curves. Of these, Sketch has Illustrator import- and export filters. Pstoedit [18], a program that attempts to convert PostScript to editable formats (see below), has export filters for Sketch and KIllustrator since version 3.13.

Most office- and consumer-oriented Windows graphics programs ignore the special capabilities of PostScript: all printing and most graphics export seem to be mediated by the Windows GDI (Graphics Device Interface), which is rather low-level. If you manage to get such low-level PostScript code converted to an editable format, you'll see tiny line fragments rather than Bézier curves; areas hatched with lines or thin slices; words chopped up into single characters. With luck the PostScript code may print, but you won't be able to do much meaningful editing.

PostScript as a programming language

PostScript is a complete programming language, with variables, conditionals and procedures. It is stack-based: a line '`1 2 add`' puts first 1, then 2 on the stack, and finally removes those two and puts their sum 3 on the stack; see figure 1. PostScript-generating applications normally start PostScript files with a fixed collection of definitions, followed by some setup, followed by the actual page descriptions, which use the earlier definitions, and end with some cleanup. See [2] for a nice, although somewhat outdated introduction to PostScript programming.

Postprocessing and DSC comments

PostScript has been designed as both an input- and an output format: a program may generate PostScript to be sent to

```
/home/siepo/gs_ntg > gs
Aladdin Ghostscript 5.50 (1998-9-16)
Copyright (C) 1998 Aladdin Enterprises, Menlo P
This software comes with NO WARRANTY: see the f
GS>1
GS<1>2
GS<2>pstack
2
1
GS<2>add
GS<1>pstack
3
GS<1>
```

Figure 1 An interactive Ghostscript session. The `pstack` command prints the operand stack to the terminal.

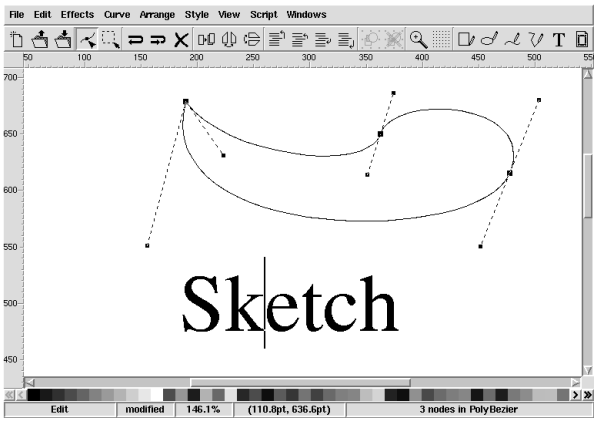


Figure 2 Sketch, a PostScript-based illustration program under Unix

a printer, but the code may also be stored in a file for subsequent postprocessing by another program. One example is incorporating a PostScript graphic into another document; another example is the psutils suite [10] which offers n-up printing and page imposition for PostScript files, i.e. arranging pages for printing in signatures.

The key to such postprocessing is use of DSC (Document Structuring Conventions) [5] comments. These comments in a PostScript file should give a postprocessing application the information it needs for doing its job without a need for parsing the actual PostScript code.

DSC comments can e.g. contain lists of required and included fonts and other resources or the number of pages, or they may mark beginning and end of sections such as prolog, setup, 'ProcSets' and font definitions.

EPS inclusion and the BoundingBox comment

Especially important for users is the boundingbox DSC comment, e.g.

```
%%BoundingBox: 12 32 400 300
```

This tells a text-processing application the dimensions of a PostScript graphic, so that it can incorporate the graphic in its own PostScript code. Above, we already saw that PostScript offers the means to position, scale and rotate the included graphic.

Such nesting only works if the PostScript file to be included satisfies some additional restrictions: it should contain only one page and it should not make certain global changes to its environment. See [6] for details. An eps- or Encapsulated PostScript file is a PostScript file satisfying these restrictions.

An eps file may also contain a screen preview. This is for the benefit of wysiwyg wordprocessing and graphics software that wants to make use of eps inclusion and needs to give the user a visual representation of the file.

There are two variants: screen previews in the form of ASCII comments, and a binary form, in which the PostScript code is preceded by a binary signature and a binary table of contents, and followed by a preview in tiff-, wmf- or Macintosh pict format. Recent versions of dvips and the LaTeX graphics package are no longer confused by preview headers. You may still want to remove them, since they are of no use to TeX and often take up a lot of space. You can do that with an editor, with a simple Perl script or, under MS Windows, with GSview, to be discussed below.

Ghostscript

Although one can do a lot with PostScript files by just reading DSC comments, it is still very useful to have a program such as GhostScript [8] that can parse PostScript:

- for printing to a non-PostScript printer
- for viewing PostScript on-screen
- for converting PostScript to other formats (pdf, bitmap formats, Adobe Illustrator)
- for creating valid eps files from generic PostScript
- for experimenting interactively with PostScript

In what follows I'll concentrate on the computer platforms I have access to, which are Win32 and Unix, or, to be specific, Windows 95 and Linux. Ghostscript is also available for VMS and for Apple Macintosh.

Printing with Ghostscript

On Unix systems, the role of Ghostscript in printing may be invisible to the user. For dvips, config.ps may contain a line `o |lpr` which tells dvips to send its output directly to the printer; lpr may use a filter:

```
#!/bin/sh
gs -dSAFER -dNOPAUSE -q -sPAPERSIZE=a4 \
-sDEVICE=cdj850 -sOutputFile=- -
```

if no real PostScript printer is installed. `cdj850` specifies the printer model; `gs -h` will display a list of available devices. Alternatively, one can print via a previewer; see below.

Under DOS and Windows, GhostScript doesn't work as transparently: generating a PostScript file and printing it with Ghostscript are two separate steps. Here is a possible batchfile for printing with GhostScript:

```
set GS_LIB=c:\gs5.50;c:\gs5.50\fonts
c:\gs5.50\gswin32c -dNOPAUSE -dBATCH
  -sPAPERSIZE=a4 -sDEVICE=cdj850 %1
```

Here, too, a command-line option `-h` will give you a list of devices.

Viewing PostScript

Although Ghostscript is capable of displaying PostScript directly, you'll get better funcionality and a nicer interface with a separate frontend. Under Unix there are Ghostview and its more modern derivative gv; under all Windows platforms there is GSview. All these can be obtained from the Ghostscript site.

These previewers read DSC comments themselves and pass PostScript code to Ghostscript. From the DSC comments they get pagination information, which enables them to jump backwards and forwards arbitrarily within a document. It also enables them to print and save ranges of pages, something which Ghostscript by itself cannot do. If DSC comments are missing, they can still display the document, but only page by page in forward direction.

Gv and GSview can optionally display the boundingbox area rather than the page area: gv has a button *BBox/Automatic/Letter/...*, GSview has a menu option *Option/EPS Clip*. GSview can add or modify a boundingbox comment (File/PS to EPS); see figure 3 although that may not be enough to turn a PostScript file into a valid eps file; see above. It can also add a preview header (*Edit/Add EPS Preview*) or remove it (*Edit/Extract EPS*).

Converting Postscript to other formats

Ghostscript supports a large number of 'devices', many of which are really file formats, such as:

- `pcxmono`, `pcxgray` and `pcx24b`, pcx bitmap format at various color depths
- `nullpage`, the null device
- `pswrite`, regularized PostScript
- `epswrite`, regularized eps
- `pdfwrite`, pdf(!)

A command

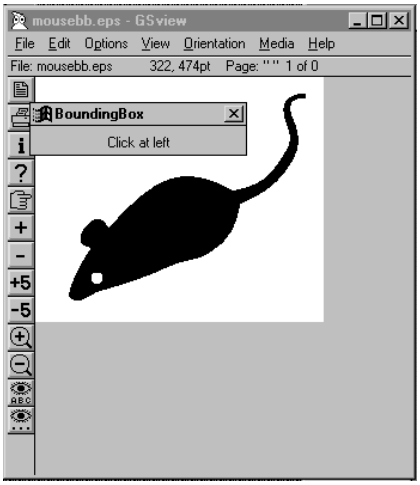


Figure 3 GSView PS to EPS: telling GSView about the boundingbox

```
gs -dBATCH -dNOPAUSE -q -sDEVICE=pcx24b \
  -sOutputFile=test.pcx test.ps
```

would convert a one-page PostScript file test.ps to an RGB bitmap test.pcx. If you want, you can add a resolution parameter, e.g. '`-r50`'. If the source is an eps graphic, you might want to run it through a script such as epstopdf by Sebastian Rahtz to make page dimensions equal to the boundingbox; see [16] and figure 6. This will ensure that Ghostscript will convert the right area. Without such a script, you can still trim the result afterwards in an image manipulation program.

Texutil from the ConTeXt package [15] contains similar functionality, based on the same source.

The pswrite and epswrite output formats are interesting because they 'draw' any text present in the file: references to characters from a font are replaced by curves drawing those characters; see figure 5. If the file is an eps graphic with just a few words of text, especially from an exotic font, this is a good way to remove font dependence.

The pdfwrite output format makes Ghostscript into an alternative for Acrobat Distiller. At the time of writing, proper support of general Type 1 fonts is still only available in beta releases, and Ghostscript will probably never have Distiller's range of prepress options. Even so, Ghostscript will in more and more cases be a perfectly adequate alternative.

Regularizing PostScript

There are various utilities, all using Ghostscript, which attempt to regularize the structure of PostScript files. The idea is to redefine primitive operators to write PostScript instructions to a file rather than to draw pixels on a device.

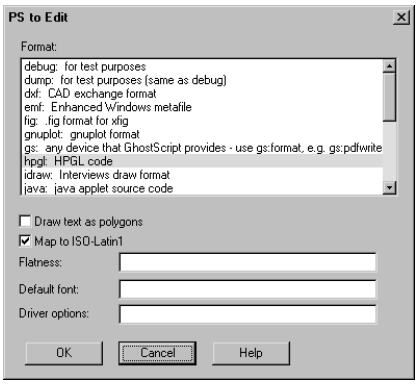


Figure 4 ps2edit as a GSView plugin; selection of output format

Ghostscript's high-level output formats mentioned above fall more or less into this category, but there are others.

Figure 5 Characters drawn as outlines, as displayed in Sketch

ps2ai.ps attempts to convert to Adobe Illustrator. It is written entirely in PostScript and is part of the GhostScript distribution. Use it as follows:

```
gs -q -dNODISPLAY -dBATCH ps2ai.ps inputfile \
  >outputfile
```

Since nowadays Illustrator can read almost arbitrary PostScript, ps2ai is more likely to be useful if you want to read a file into another PostScript-based drawing program.

ps_conv [17] also tries to make generic PostScript Illustrator-compatible and works in a similar way, except that a script or batch program has to add a header part and a tail. Contrary to ps2ai, it draws characters rather than leave them in text form; see figure 5 for an example of characters drawn as outlines.

pstoedit [18] converts to a variety of vector formats, including regularized PostScript, xfig, gnuplot, hpgl and nowadays also Sketch and Killustrator. Other formats, such as wmf, can be supported via plug-ins, and it will run ps2ai for you if you want.

Pstoedit also works by redefining primitives. It is a binary executable and calls Ghostscript – or another PostScript interpreter – to read the PostScript code. On the windows platform, it can be installed as a plug-in to GSview version 2.71 or later; see figure 4.

```
$bbneeded=1;
$bbpatt="[0-9\.\-]";
while (<>) {
 if (
  /%%BoundingBox:(\s$bbpatt+)\s($bbpatt+)\s($bbpatt+)\s($bbpatt+)/
 ) {
     if ($bbneeded) {
      $width = $3 - $1;
      $height = $4 - $2;
      $xoffset = 0 - $1;
      $yoffset = 0 - $2;
      print "%%BoundingBox: 0 0 $width $height\n";
      print "<< /PageSize [$width $height] >> setpagedevice\n";
      print "gsave $xoffset $yoffset translate\n";
      $bbneeded=0;
     }
 }
else
  {    print;  }
}
print "grestore\n";
```

Figure 6 Perl code by Sebastian Rahtz giving an eps file a page definition matching its bounding box; a more complete version can be downloaded from [16].

All these conversion options are less of an overkill than you might think: none of them does a perfect job, and sometimes it helps to apply two of them in succession.

Pdf in prepress

Over the last few years, Adobe has pushed pdf (Portable Document Format, see [7]) as a 'final-form' format for distributing documents. Pdf is a subset of PostScript level 3 without programming functionality, and with page independence. Pdfmarks are unique to pdf. The restrictions of page independence and lack of programmability should make pdf more tractable than generic PostScript. The graphics industry has enthusiastically adopted pdf as a workflow format.

Designers, desktop publishers and printers have been very reluctant to hand off or receive printjobs in the form of PostScript dumps, and usually preferred source files. Of course, this created lots of problems with missing files and version differences and text reflow, but apparently these problems were preferable over dealing directly with PostScript files. One has to admit that there was a point in handoff of source files: it certainly facilitated last-minute fixes, and printer/imagesetter settings could be selected by the person running the printer/imagesetter.

Professional graphics software increasingly supports pdf in terms of direct pdf generation and in terms of reading pdf. Various modern TeX implementations (Web2c, VTeX) include a pdf backend, which supports pdf inclusion as a replacement for eps inclusion (VTeX can parse PostScript and is therefore capable to include eps files directly).

Software for pdf

As to generating pdf from PostScript: we already saw that Ghostscript can do that. For best results, however, Acrobat Distiller is still to be preferred. Distiller is available for the Windows, Macintosh and some Unix platforms. Linux users may be interested to know that Distiller 3 runs well under Linux and Wine ([19]).

Adobe's free *Acrobat reader* is available for the Windows- and Macintosh platforms, and for a variety of Unix platforms, including Linux. Ghostscript and its frontends also support viewing pdf.

xpdf [20] is a non-commercial specialized pdf viewer, available under Unix and VMS. It is mainly of interest for some of the accompanying utilities – also available on the Win32 platform: *pdftotext* extracts text from a pdf and *pdfimages* the bitmaps. Of course, you can let Ghostscript convert PostScript or pdf to a bitmap, but original bitmapped data will usually get resampled, resulting in loss of quality. pdfimages will recreate the original bitmaps pixel for pixel, although it won't respect the original color depth. Since Ghostscript can convert eps to pdf, *it is finally possible to convert bitmapped eps files without resampling back into an editable format!*

For the Windows- and Macintosh platforms, reasonably-priced commercial utilities have arrived to manipulate pdf, e.g. a visual editor Enfocus Pitstop [21], or Quite a Box of Tricks [22] which can do some very useful conversions, such as converting pages to grayscale. Color separation- and imposition software now becomes available as plugins for Exchange. Check out `www.pdfzone.com` for more. Final printer settings now can be made when printing from Exchange; the settings made when the original PostScript file was generated can usually be overruled[1]. The Crackerjack Exchange [23] plug-in gives access to high-end prepress functions.

Pdf is also interesting because of its interactive possibilities, but these fall outside the context of this paper.

References

[1] PostScript Language Reference Manual, Adobe Systems Incorporated, Addison Wesley, 1999

[2] PostScript Language Tutorial and Cookbook, Adobe Systems Incorporated, Addison Wesley, 1986.

[3] Adobe web site `www.adobe.com`

[4] Adobe developer information, `partners.adobe.com /asn/developer/main.html`

[5] PostScript Language Document Structuring Conventions Specification Version 3.0, available from [4]

[6] Encapsulated PostScript File Format Specification Version 3.0, available from [4]

[7] Portable Document Format Reference Manual Version 1.3, available from [4]

[8] Ghostscript, `www.cs.wisc.edu/~ghost`. Viewer frontends can also be obtained via this url.

[9] Ghostscript User Manual, Thomas Merz, 1997, `www.ifconnection.de/~tm`

[10] Psutils by Angus Duggan, `ftp://ftp.dcs.ed.ac.uk/pub/ajcd` and `ftp://ftp.tardis.ed.ac.uk /users/ajcd`

[11] Mayura Draw, `www.mayura.com`

[12] Sketch, `www.online.de/home/sketch`

[13] Killustrator, `wwwiti.cs.uni-magdeburg.de/~sattler/killustrator.html`

[14] GYVE, the GNU Yellow Vector Editor, `www.maru.cs.ritsumei.ac.jp/~sanchan/gyve`

[15] ConTeXt, `www.pragma-ade.nl`

[16] `www.tug.org/applications/pdftex/epstopdf`, a Perl script by Sebastian Rahtz to set the page dimensions of an epsfile to the bounding box.

[17] Various PostScript tools by Bogusław Jackowski, Piotr Pianowski and Piotr Strzelczyk, `ftp://ftp.GUST.org.pl/pub/TeX/GUST/contrib/PS`; also available from CTAN

[18] Pstoedit, `www.geocities.com/SiliconValley /Network/1958/pstoedit`

[19] Wine Is Not an Emulator, `www.winehq.com`

[20] Xpdf, `www.foolabs.com/xpdf`

[21] Enfocus Pitstop, `www.enfocus.com`

[22] Quite a Box of Tricks, `www.quite.com`

[23] Crackerjack, `www.lantanarips.com`

[24] Pdfzone for information on pdf software, both free and commercial, `www.pdfzone.com`

1. This is not true for the resolution of bitmap fonts generated by dvips. Acrobat handles Type1 fonts much better than bitmapped fonts so the latter should be avoided if at all possible. It is still possible to print from Acrobat with bitmapped fonts at the wrong resolution, but the results won't be optimal.