bestuur

Van de voorzitter

Erik Frambach
Rijksuniversiteit Groningen
email: E.H.M.Frambach@eco.rug.nl

MAPS Nieuwe Stijl

De eerste echte 'MAPS Nieuwe Stijl' is uit. Nieuw aan die stijl is echter niet de vormgeving of stijl, maar juist de inhoud. Van oorsprong betekende MAPS 'Minutes & Appendices', maar die zijn nu losgekoppeld. De pure NTG-zaken (de 'Minutes') worden nu alleen nog aan NTG-leden toegestuurd, terwijl de artikelen (de 'Appendices') netjes bij elkaar in de MAPS worden gepubliceerd.

'MAPS' is daarmee niet langer een acroniem, maar een naam. En niet zo maar een naam, maar een naam met een reputatie. Sinds het ontstaan van de NTG is er hard aan gewerkt om de NTG-leden twee keer per jaar te tracteren op een verzameling van interessante artikelen over van alles wat met TEX te maken heeft. Hoogtepunten daarin zijn ongetwijfeld de proceedings van de EuroTEX conferentie in Arnhem (1995), verschillende 'MAPS specials', en niet te vergeten het jubileumnummer dat vorig jaar is uitgegeven.

Daar mogen we best trots op zijn, en ik hoop dat we dit niveau nog lang vol kunnen houden. Uiteraard gaat daar veel tijd en energie in zitten. Daarom is het van levensbelang voor de NTG dat de leden zich actief inzetten om de MAPS met interessante artikelen te vullen. Dat kan door zelf te schrijven over ervaringen, of door de redactie attent te maken op belangwekkende gebeurtenissen, goede artikelen die elders zijn gepubliceerd, of door zelf redactiewerk te gaan doen. Alle hulp is welkom.

TEX Live 4

Versie 4 van de befaamde 'TEX Live' cdrom komt er aan. Deze cdrom bevat de nieuwste versie van Web2c (7.3) voor een keur aan operating systems, waaronder Unix (Linux, HP, DEC, IBM, SGI, SUN), Amiga en MS-Windows. Het lijkt erop dat de TEX wereld langzamerhand naar één systeem toegroeit dat overal werkt, namelijk Web2c. Dat is een goede zaak omdat op die manier een hoop onderhoud centraal kan plaatsvinden, en het wiel niet telkens opnieuw hoeft te worden uitgevonden. Voor gebruikers betekent het dat ze betere support zullen krijgen. Ook aardig om te weten is dat Web2c 7.3 een nieuwe mogelijkheid biedt om de *input encoding* van bestanden te respecteren. Hoe dat precies werkt zal ik hier niet uitleggen, maar het zal duidelijk zijn dat dit erg prettig is voor wie schrijft in bv. oost-Europese of niet-westerse talen. Verder is vermeldenswaardig dat ε-TEX en PDFTEX nu geïntegreerd zijn. Eén executable die *alles* kan: dat opent perspectieven!

Minstens zo belangrijk is de TDS (TEX Directory Structure) die met TEX Live wordt meegeleverd. Die boom kan zonder meer als referentie worden beschouwd. Een completere boom vind je nergens.

Kortom, we zien hier een ontwikkeling in de richting van een meer uniforme TEX-configuratie die voor (bijna) de hele wereld bruikbaar is.

4TEX 5

Van 4TEX, een van de meest gebruikte systemen onder MS-DOS en MS-Windows, is inmiddels een gloednieuwe Windows-versie uitgekomen. De nieuwe versie heeft een geheel gemoderniseerde interface met de bekende Windows 'look & feel'. Bovendien zitten er snufjes in die nog nooit eerder vertoond zijn. Uiteraard is de motor van de nieuwe 4TEX ook Web2c, zodat die moeiteloos kan samenwerken met TEX Live 4. Ook de documentatie van 4TEX is geheel vernieuwd, en completer dan ooit. Aangezien de documentatie een uitgebreide beschrijving bevat van alle Web2c programma's, is die zelfs voor Unix-gebruikers en andere Windows-haters interessant.

Conferenties

Ook dit jaar kunnen we twee grote TEX conferenties verwachten. De TUG-conferentie vindt dit jaar plaats in Vancouver, Canada, van 15 tot en met 19 augustus. Het thema is: 'TEX Online: Untangling the Web and TEX'. Op de WWW-pagina http://www.tug.org/tug99/ is meer te lezen over de conferentie.

De EuroTEX-conferentie vindt dit jaar plaats in Heidelberg, Duitsland, van 20 tot en met 23 september. Het thema is daar: 'Publizieren von Dokumenten'. Zie de WWW-pagina http://www.dante.de/eurotex99/ voor meer informatie.

Gebruikersgroepen

Dit keer zijn er twee nieuwe gebruikersgroepen te verwelkomen. TUG-Philippines (Filippijnen) en HunTEX (Hongarije). We wensen hun veel succes!

Redactioneel

Taco Hoekwater
Siep Kroonenberg

Voor u ligt ondertussen al weer MAPS #22. Zoals beloofd is dit een themanummer, bijna de helft van deze MAPS bevat artikelen die op een of andere manier over fonts gaan. Maar dat vult nog geen hele MAPS (net een halve) en daarom ook een vrij grote collectie artikelen over volledig andere zaken.

De afgelopen paar maanden heeft de redaktie niet stilgezeten. De bibliografie op het net is nu weer helemaal bijgewerkt (zie `http://www.ntg.nl/maps/electromaps.html`), en alle PDF bestanden van de MAPS-en tot en met nummer 20 zijn te downloaden. In de komende tijd gaan we ook de web-pagina's herstylen, daarover in de volgende MAPS meer.

Even wat over de vormgeving van deze MAPS: er zijn weer eens wat kleine veranderingen. Zo zijn de titels in de tweekoloms layout terug verschoven naar links, wat voor een wat rustiger beeld zorgt. Het font van de kopjes is een heel klein beetje kleiner geworden, en we hebben wat finetuning gedaan aan de witruimte in de openingen.

Nu eigenlijk de hele maps uit Appendices bestaat, had het ook niet zo veel zin meer om alles maar 'Bijlage' te blijven noemen. Daarom wordt het kopje boven de titel nu gebruikt om een indicatie te geven van de inhoud van het artikel.

Natuurlijk begint deze MAPS weer met de min of meer vaste onderdelen zoals de agenda, het overzicht van de mailing lijsten en de sectie 'Gezeefd van tex-nl'.

Na de artikelen over fonts begint dan een interessante sectie met artikelen over een veelheid aan onderwerpen: van het jaar 2000 tot aan boekbesprekingen. Veel aandacht voor 'Nederlandse' ontwikkelingen hier (4spell en 4project, eetex, context, mkpic) en veel informatie over PDF. Nu de MAPS altijd van PDF gedrukt wordt zijn deze artikelen in ieder geval voor de redactie al interessant leesvoer.

Onze NTG voorjaarsbijeenkomst gaat over de 'Toolbox' van programma's rond TEX. In navolging van de bijeenkomst zal de volgende MAPS dan ook weer een themanummer worden. Daarom in deze MAPS helemaal geen 'Toolbox': die bewaren we voor de volgende keer.

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

De gebruikte fonts zijn Adobe Times-Roman (met expert set) en Frutiger, met een speciaal versmalde Courier voor de 'verbatim' omgevingen. Eventuele wiskundige formules worden gezet met de MathTime fonts van Y&Y.

Compilatie gebeurt met web2c versie 7.2, grotendeels onder Linux. We genereren PDF voor de drukker, deels rechtstreeks met pdftex 0.13a, deels via dvips versie 5.78 en Adobe Distiller 3.02. Voor het verwerken van plaatjes en voor Distiller zijn we nog steeds afhankelijk van het Windows platform; we hopen echter vurig dat die afhankelijkheid een snel aflopende zaak is, temeer daar blijkt dat de MAPS de capaciteiten van Exchange af en toe kennelijk te boven gaat.

De MAPS wordt direct vanaf PDF gedrukt op 90-grams coated papier op een resolutie van 2400 dpi in een oplage van 350 stuks.

Het versturen gebeurt deels met de ptt partijpost, deels door de internationale bezorging van Kluwer Academic Publishers (waarvoor onze hartelijke dank) en deels intern door de buitenlandse gebruikersgroepen met NTG leden.

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`

FGBBS – NTG's Bulletin Board

AllTEX – TEX, LATEX, etcTEX

ltxiii – LATEX 3.0

4TEX – Het volledige TEX *runtime systeem* voor MS-DOS PC systeem, gebaseerd op emTEX en 4DOS.

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

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 de administratie- en verzendproblemen 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 kontakt op te nemen met het secretariaat van de NTG.

TEX kalender 1999

- TUG conferentie 15–19 augustus in Canada, Vancouver
- EuroTEX conferentie 20–23 september in Heidelberg, Duitsland
- BachoTeX'99 (GUST conferentie) 1–3 mei in Bachotek, Polen

MAPS 99.2

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

*1 juli '99 (*MAPS *99.2; #23)*
*15 januari 2000 (*MAPS *2000.1; #24)*

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

- *Bij voorkeur* in gebruikmakend van de LATEX 2_ε class file `maps.cls` of de ConTEXt module `m-map-01`. Beide files zijn via de redactie te verkrijgen en beschikbaar op de TEX-NL 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 ongeformatteerd.
- Plaatjes bij voorkeur als (Encapsulated) PostScript file plus het oorspronkelijke formaat; dit laatste om eventuele problemen beter te kunnen oplossen.

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.

STANFORD UNIVERSITY STANFORD, CALIFORNIA 94305-9045

DONALD E. KNUTH
Professor Emeritus of The Art of
Computer Programming
Computer Science Department – Gates 4B
Telephone [650] 723-4367

August 28, 1998

Professor Siep Kroonenberg
Faculteit der Economische Wetenschuppen
Rijkuniversiteit Groningen
The Netherlands

Dear Professor Kroonenberg,

Bravo for the new NTG logo. I fell in love with it *immediately* upon seeing the cover of MAPS number 20! Great possibilities for color too.

Sincerely,

Donald E. Knuth
Professor

DEK/pw

P.S. And what a fantastic issue of MAPS . . . it has encouraged me to purchase a good Dutch dictionary.

fgbbs

Bericht van voorheen FGBBS — Voorjaar 1999

Frans Goddijn
frans@iaf.nl

abstract

Jammer genoeg weet ik niet meer wanneer het FGBBS precies ‘de lucht in’ is gegaan, of beter gezegd ‘de telefoonlijn op’. Het is denk ik begin 1993 geweest. Een 14k4 modem heette toen nog een *high-speed* modem, dus dat is, in computertijd gemeten, lichtjaren geleden. En nu het BBS niet meer bestaat, ben ik vergeten op welke dag ik de stekker eruit trok. . .

Waarom?

Er was een goede reden om te stoppen, maar dat was niet *mijn* reden. Het is wel zo dat het Internet voor elke pc-bezitter met een dubbelklik bereikbaar is geworden, en dat daarmee de functie van FGBBS veranderde van ‘outpost’ voor TeX-zendingswerk in een antikwariaat voor oude stylefiles en formats uit de stone age.

Voor mij was de belangrijkste reden: stilte. Nadat zes jaar lang op de ‘werkkamer’ thuis (waar het gezin ook tv kijkt) de een na de andere pc vierentwintig uur per dag, alle dagen van het jaar bijna onafgebroken had aangestaan, merkte ik dat de stilte die inviel als hij eventjes uit stond bijna verpletterend was. Hoe geruisarm ook de ventilatoren (en later ook nog de cpu coolers) waren, altijd hing er een ‘white noise’ in de kamer die pas opviel als hij af en toe even wegviel. De ventilatoren waren begin jaren ’90 nog erg lawaaierig, en nadat ze een paar maanden hadden gedraaid, werden ze ongemerkt steeds lawaaieriger.

Toen ik de knoppen van FGBBS omdraaide en de stilte niet meer tijdelijk was, bleek de stilte bijna tastbaar, als koel roomwit marmer. De stilte was ook diep, zo diep dat ik er zowat hoogtevrees van kreeg. . .

Aanleiding

De directe aanleiding voor het laten ‘inslapen’ van het BBS was de overstap van OS/2 Warp4 naar WindowsNT Workstation 4. De installatie van alle verschillende BBS programma’s (meerdere werkten samen om de gebruiker een gevarieerde user interface te bieden) was onder OS/2 niet simpel geweest, maar destijds had ik Henk de Haan als actieve cosysop. Hij woonde weliswaar in Delft en ik in Arnhem, maar we belden veel en we deelden samen de benodigde hardware. Hij was bezig aan zijn proefschrift op het gebied van kernfysica maar had de neiging om in plaats daarvan aan andere dingen te werken, hoe moeilijker en uitdagender hoe leuker. Van die werklust en dat escapisme kon ik dankbaar profiteren! Nu heeft Henk echter al lang geleden zijn proefschrift over door muonen gecatalyserde kernfusie gepubliceerd en hij doet onnavolgbare dingen met ontwerpen van smart cards.

OS/2 is nu wel verder ontwikkeld, maar nieuwe versies van voor mij belangrijke programma’s wilden onder OS/2 niet meer goed draaien (administratieve dingen als telebanking en boekhouding, maar ook PERL, PDFtex). Voor die dingen was het dringend gewenst dat ik naar een Windows platform over ging. Ik ben nog steeds overtuigd van de superioriteit van OS/2, maar ik heb nu eenmaal weinig aan een supersnel en superstabiel systeem waarop geen bruikbare software meer draait.

Onder NT echter, zo verwachtte ik, zou een goed deel van de FGBBS pogrammatuur problematisch werken en om mezelf de frustratie en de gebruikers een onaangename verrassing te besparen, ben ik eind 1998 al begonnen met het voorzichtig afbouwen van het systeem. Diverse mensen die via FGBBS een emailadres hadden, konden zo tijdig omschakelen naar een account bij een echte Internet Provider. Toen de nieuwe NT machine kwam, kostte het een dag werk om zoveel mogelijk FGBBS spullen aan de praat te krijgen en daarna heb ik die stukje bij beetje, ordelijk, afgebroken, telkens als er een onderdeel gemist kon worden. Nog een hele gedoe, want alles was met elkaar verknoopt en ik wilde het gebouw voorzichtig ontmantelen. Een explosief onderaan de hoge toren was ongewenst.

Inmiddels haalt niemand meer post op via FGBBS. Een paar laatste onderdelen van het systeem onderhouden nog een paar mailinglijsten, onder andere die voor de besteling van 4TEX cd’s, maar ook dat zal een dezer dagen worden verhuisd naar de NTG server. Een deel van die software is niet klaar voor 2000, maar daar zit ik niet over in. Eer het zover is, zijn ze gewist.

Deleten zonder sproedel

Het wissen van de vele honderden megabytes aan TeX materiaal dat destijds, vooral door Henk de Haan, is verzameld, geordend en van toelichting voorzien, ging onthutsend snel. Ik keek om me heen terwijl de harddisk met een zacht knisperend geluid de bestanden vermaalde. Zou er

misschien iemand een woordje willen spreken? Maar er was niemand, net zo min als we destijds champagne hebben geschonken of historische woorden gesproken toen het systeem begon te werken.

Ik dacht wel aan Jason Fesler, de programmeur van de WME user interface, een slimme manier van schermbesturing op BBS'en. Henk en ik waren destijds de 64e gebruiker die het registreerden. Jason heeft nog eens van een Arnhemse weldoener (niet van FGBBS) een werkbeurs gekregen om een nieuwe versie van WME te produceren waarmee veel geld te verdienen zou zijn, maar die is nooit afgekomen en nu bezit de stille investeerder daarvoor in de plaats een hoeveelheid programmacode waar hij nooit wat mee zal doen, omdat hij niet weet wat het allemaal doet. Jason Fesler schreef wel GIGO (garbage in, garbage out), het programma dat Internet email vertaalt voor gebruikers van het BBS en daarvan waren we wereldwijd de dertiende betalende klant. Ook GIGO bracht voor het gezin Fesler niet voldoende op, en nu is hij systeembeheerder voor een lokale provider. GIGO draait nog ten behoeve van de laatste mailinglists, maar WME is spoorloos van de schijf gewist. Van FrontDoor, het programma dat ooit alle andere aanstuurde, heb ik nog wel een nieuwe versie in huis gehaald, maar toen daarmee wat kleine problemen ontstonden dacht ik: *hey, why bother*, ik start het gewoon nooit meer op.

Misschien was het wel leuk geweest om hier tot slot te melden hoeveel bezoekers het FGBBS in die jaren heeft gehad en hoeveel megabytes er zijn opgehaald. De logfiles waren er, maar ik ben al lang vergeten hoe die superslimme 4DOS scripts van Henk de Haan werken, die zulke massa's data konden verwerken tot leuke statistieken. Hoe dan ook, het is over, en daar gaat het om. Weg is weg. Wel wil ik graag de NTG danken voor de support, die diverse jaren in de vorm van subsidie aan het FGBBS is gegeven.

Plezier met NT

Toch staat de pc nog steeds aan maar dan overdag. Direct nadat ik op ben gestaan, verjaag ik de stilte met de white noise die de hard disks, cpu cooler en ventilator samen maken en ik haal mijn email op. Die lees ik terwijl ik me aankleed. Na het ontbijt schuif ik aan het bureau en dan begint een lange dag, vaak met plezier maar ook met veel gekerm en gekreun, en licht panische telefoontjes naar slimmere computervrienden. Want in de afgelopen eerste maand met NT heb ik eindelijk PERL perfect aan het werk gezien, bijvoorbeeld bij het met PDFtex compileren van ConTeXt files door middel van het texexec.pl script, maar eer die dingen geïnstalleerd waren... brr. Ik voel me af en toe tussen al die nieuwe features en nieuwe gevaren als een geblinddoekte, die moet rennen voor zijn leven in een zuilengalerij. Gelukkig zijn er trouwe TeX vrienden, die inmiddels weten hoe laat het is als ik bel: 'hallo, heb je even tijd voor een vraagje?'...

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. Elders in deze MAPS staat een bijdrage van Piet van Oostrum over de web-pagina's van de NTG. 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 eenvoudig 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 discussie lijst over TEX in Nederland.
4tex Lijst voor vragen en mededelingen betreffende het 4TeX-systeem van Erik Frambach en Wietse Dol.
ntg-sgml Discussielijst van de SGML-werkgroep binnen de NTG.
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 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, ntg-smgl 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).

GUST

Announcing BachoTEX'99

Erik Frambach
University of Groningen
email: E.H.M.Frambach@eco.rug.nl

abstract
An announcement of GUST's annual conference in Bachotek, Poland.

keywords
Bachotek, BachoTEX'99, GUST, conference

GUST

Many years ago it was Kees van der Laan who set up contacts with the Polish TEX users group GUST. These contacts flourished ever since, resulting in cooperation, friendship, mutual invitations to tutorials and conferences. And let's not forget the EuroTEX bus that enabled many Polish people to attend the EuroTEX'95 conference in Arnhem.

I've had the pleasure of attending the EuroTEX'94 conference in Gdańsk, and I was impressed by the Polish hospitality and friendliness. Last year GUST made another great effort to the benefit of the TEX community by organizing the TUG'99 conference in Toruń. Naturally that was another great success.

But that's not all. GUST also has its own annual meeting. There is a small village called Bachotek, which is surrounded by forests and lakes, where this event takes place every Spring. Naturally the event is called 'BachoTEX' for the occasion. In several MAPS issues you can find reports on BachoTEX conferences.

BachoTEX

Actually this is not a real conference center but rather a vacation resort with some conferencing facilities. For three days you can live in the woods in wooden houses near a lake. Plenty of opportunities for hiking, fishing, swimming (?), snake watching, or just enjoying the peaceful atmosphere and the clean air. Is it any wonder that many GUST members bring their whole family to the conference?

And oh yes, there are many lectures, tutorials and discussions on TEX-related issues. It's almost a EuroTEX meeting because you can expect to meet people from Poland, Germany, The Netherlands, Great Britain and Hungary, perhaps more. So don't worry too much about language problems. Lectures are in general in Polish, but you can ask any English speaking Polish friend to translate for you.

The program

This year the conference will take place from 1–3 May, and this year's theme is *Practical aspects of electronic publishing*. As the theme suggests, the conference will focus on practical aspects of the production of all sorts of electronic publications. The list of possible topics includes:

- TEX, macros, drivers, etc.
- multilingual publications
- graphics and fonts
- PostScript and PDF
- standards: SGML, HTML, XML, MathML

As always, the conference program will be completed with *tutorials* covering wide range of interesting subjects presented by renowned TEX experts. The conference fee will be announced shortly but as usual it will be approximately 100 USD. Check out `http://www.gust.org.pl/BachoTeX/99-e.html` for details on the conference.

It looks like this year BachoTEX will be visited by more Dutch TEXies than usual. Obviously the conference in Toruń was good promotion for GUST.

If you need an opportunity to combine useful learning experience in TEX c.s. *and* a good time to relax and have fun with friends, then to BachoTEX you should come. Bring your whole family!

And don't forget to register for the tutorial on fire-spitting (not a joke). If that's too much for you, just make sure you don't miss the bonfires.

TUG’98, Toruń, Polen

Foto’s van Luzia Dietsche en Gyöngyi Bujdosó;
samenstelling Siep Kroonenberg

Linksboven: Kasteel Golub-Dobrzyn, Gilbert van den Dobbelsteen aan het vuurspuwen [LD]. *Rechts, van boven naar onder:* Golub-Dobrzyn, beest aan het spit [LD]; campus Copernicus Universiteit, gezang met guitaarspel van Bogusław Jackowski [GB]; Gilbert, Hans en Taco aan het hacken in de campus-disco [LD]. *Onder, van links naar rechts:* concert in het Artus gebouw in Torun [GB]; TEX leeuw [LD]; dansen in het Artus gebouw [GB].

Frans Goddijn
frans@iaf.nl

tex-nl 5

Gezeefd uit de TEX-NL discussie-lijst

abstract
Dit is het vijfde deel uit een serie die in 1993 door Philippe Vanoverbeke in de MAPS is gestart. Philippe maakte een selectie van berichten uit de TEXNL lijst. Een aantal oplossingen, hints en gouden tips over onderwerpen waarvan je weleens denkt: „hoe zat dat ook alweer?" In de MAPS van voorjaar 1998 stond deel 4. Voor deze aflevering is een aantal berichten gezeefd uit TEXNL van maart 1998 tot januari 1999. Met dank aan Piet van Oostrum die opnieuw de meeste antwoorden aandroeg, en aan de andere actieve abonnees van TEXNL!

keywords
TEXNL, Oostrum, guru, tips, hints, hack, bug

Titels

(We beginnen met de alinea die waarmee de vorige aflevering van deze serie onverwacht eindigde . . .)

Is het mogelijk om i.p.v. op alle pagina's de hoofdstuktitel te hebben de titel van de `\section` te bekomen door gebruik te maken van deze opmaakodes?

```
\renewcommand{\chaptermark}[1]{}
\renewcommand{\sectionmark}[1]{\markright{\thesection\ #1}}
\fancyhf{}
\fancyhead[LE,RO]{\bfseries\thepage}
\fancyhead[LO,RE]{\bfseries\rightmark}
```

Of, als section en subsection titels gewenst zijn:

```
\renewcommand{\chaptermark}[1]{}
\renewcommand{\sectionmark}[1]{\markboth{#1}{}}
\renewcommand{\subsectionmark}[1]{\markright{\thesection\ #1}}
\fancyhf{}
\fancyhead[LE,RO]{\bfseries\thepage}
\fancyhead[LO]{\bfseries\rightmark}
\fancyhead[RE]{\bfseries\leftmark}
```

Grotere inhoudspagina

Voor een tijdelijk grotere pagina gebruik je `\enlargethispage*{1cm}`

Hoe doe ik dat voor een pagina met inhoudsopgave?

```
\addtocontents{toc}{\protect\enlargethispage*{1cm}}
```

Regelafstand in tabular

Een tip van Herman Haverkort om simpel een grotere regelafstand te krijgen binnen de tabular omgeving: `\arraystretch`

Voorbeeld:

```
\documentclass[11pt]{artikel3}
\begin{document}
\def\arraystretch{2}
\begin{tabular}{|l|l|}\hline
tra & la\\\hline
tra & la\\\hline
tra & la\\\hline
tra & la\\\hline
\end{tabular}
\end{document}
```

Grotere inhoudspagina

`\addtocontents{toc}{\enlargethispage*{1cm}}` bleek niet te werken, het gaf een onnavolgbare foutmelding over `Argument of \@sect`.

De oplossing bleek:

```
\addtocontents{toc}{\protect\enlargethispage*{1cm}}
```

Hoe kom je aan een trademark-teken?

```
\def\omcirkeld#1%
  {\raise1ex\hbox{\ooalign{\hfil\raise0.07ex\hbox
    {{\textsf{\scriptsize#1}}}\hfil\crcr\mathhexbox20D}}}
```

overigens kan het ook zo:

```
\documentclass{article}
\usepackage{textcomp} % hier zitten een heleboel handige symbolen in
\begin{document}
textcomp\textsuperscript{\textregistered}
\end{document}
```

Brede tekst boven dubbelkolom

Twocolumn is leuk, maar hoe kan ik nu zorgen dat ik nog een tekst in de breedte boven de twee kolommen krijg, en dan ook nog op dezelfde pagina? ;-) Ik dacht eerst „ik zet er gewoon wat later in de tekst `\twocolumn` meer”, maar dat werkt dus niet.

```
\twocolumn[Tekst over de hele breedte]
```

Als je een abstract en dergelijke daar wilt hebben, dan moet je nog wat meer uitspoken (geleend van o.a Donald Arseneau).

```
\twocolumn[{
\begin{@twocolumnfalse}

  \maketitle
  \begin{abstract}
  ...
  \end{abstract}

\end{@twocolumnfalse}
}]
```

Voorwoord en nawoord niet nummeren

In `\documentclass{article}` met `\usepackage[dutch]{babel}` wil ik een ongenummerde sectie Voorwoord en een ongenummerde sectie Nawoord hebben. Beide secties moeten echter wel in de `\tableofcontents` voorkomen.

```
01 \setcounter{secnumdepth} {0}
02 \section{Voorwoord}
03 \addtocounter{secnumdepth} {3}
```

`secnumdepth` geeft aan hoe diep je nummert: sectie + paragraaf + subparagraaf = 3. Wanneer `secnumdepth` 0 is komt er geen nummer te staan. Echter de sectie wordt wel opgenomen in de `tableofcontents`. In 2.3.1 van The LaTeX Companion (8th printing, August 1997) staat meer over dit onderwerp.

Groter font

Hoe kan ik mijn font nog groter maken dan `\Huge`? Ik heb nu een kop boven een tekst staan, maar die is zo afschuwelijk klein, dat ziet er niet uit (Ik gebruik bij die kop trouwens `\Huge\sc`)

```
\font\test=cmr10 scaled 5000
\font\testtwo=ptmr7t scaled 5000
```

Dit produceert bv. 50-punts cmr10 en Times

Hoe begin in aan ConTeXt en PDFtex?

De manuals van het veelbelovende pakket ConTeXt gaan ervan uit dat je al een werkende installatie hebt waarop het ConTeXt format kan worden gegenereerd, en ook gaan de manuals ervan uit dat je PDFtex goed draaiende hebt. Straks heeft iedereen dat vanzelf bij all-in apkketten als de nieuwste 4TEX of de TeX live installatie, nu is het voor sommigen nog een heel gefröbel voor de eerste glorieuze schreden op het ConTeXt pad gezet kunnen worden. Hieronder het verslag van een installatie. Let wel op: inmiddels zullen er zeker veranderingen zijn geweest in de aangegeven vindplaatsen en in details van de installatie.

This is a small report on what I have done to set up a compact but powerful combination of CONTEXT and PDFTEX on my system.

They enable me to get the remarkable results that several CONTEXT articles and lectures have shown me and which have tickled my appetite for CONTEXT, and they enable me to produce high quality PDF output In this report I explain where I got the necessary files, where I put them on my system, and what I did to get it running.

I am aware that preferably this task were done by an expert, so as to get it all fool proof, eloquent and smart looking. I merely did my best to make sure that the help I sought and received from Hans Hagen could be used for the benefit of others as well...
Good luck and enjoy where you can! I got the files `pdftex.zip` and `pdftexlib-0.12.zip` from:

`ftp://ftp.cstug.cz/pub/tex/local/cstug/thanh/pdftex-testing/pdftex-0.12n/`

```
pdftex.zip          executables
pdftexlib-0.12.zip  platform-independant files
                    (I renamed this file pdftex-1.zip)
```

I created a directory CONT-PDF on my F: drive and installed the following files there, mostly taken from the abovementioned 2 downloaded files:

```
cont-nl.fmt   2487620  15-06-98  13:05 CONTEXT format file, see below
pdftex.cfg        119  15-06-98  14:18 only a few lines, see below
pdftex.exe     601600  30-05-98  20:32 executable from pdftex.zip
pdftex.poo      30387  30-05-98  20:23 came with pdftex.zip
pdftex.rea       5609  15-06-98  13:38 readme file from pdftex.zip
texmf.cnf       16002  15-06-98  14:16 configfile, see below
ttf2afm.exe     76800  30-05-98  20:32 executable from pdftex.zip
```

The file pdftex.cfg contains these lines

```
output_format 0
compress_level 9
decimal_digits 3
page_width 210mm
page_height 297mm
horigin 1in
map pdftex.map
```

In a second directory, CONTEXT, I put and unzipped the following files: CONTTEX.ZIP and ALLES.ZIP (both available via `http://www.ntg.nl/context/`)

Then I made sure that these new directories were in my PATH command, for example:

```
PATH=...f:\cont-pdf\;f:\context;
```

(the '...' stands for all other stuff on this line in my PC)

Also, in my `autoexec.bat` I put the following line:

```
SET TEXMFCNF=F:/CONT-PDF/
```

The use of this line is that PDFTEX will know where to find the file TEXMF.CNF

This TEXMF.CNF needs editing. See far below for my version of it.

Mostly it's commented-out text explaining about spiffy options, and at the low end there are a number of relevant lines which need to be congruent with the paths on your system. For instance, the TEXMF.CNF has lines that must point to the correct CONT-PDF directory which you've just created and also it must point to the well-known directories for TeX applications, like TFM's, Postscript fonts etcetera. What I did was use the information of my working 4TEX setup to point towards the several directories. If in severe doubt, let a line point to your drive with TeX stuff with all subsequent directories on it, like "F://" which makes the system work while making it hooribly slow.

Now, on a DOS prompt, with the PATH command and SET variable checked, I went to the CONT-PDF directory and gave the following command:

```
pdftex --ini cont-nl
```

which started off PDFTEX.EXE initiating and creating a CONT-NL.FMT tex format file in the same directory. All CONTEXT sources were used from the CONTEXT directory.

Then, with a test.tex file containing for simplicity only these lines

```
\gebruikspecials[tpd]
\starttekst
Now will this work or won't it operate?
\stoptekst
```

you can, in the CONT-PDF directory, type:

```
pdftex &cont-nl test.tex
```

which yields a PDF file of the given test. Without the line saying `\gebruikspecials[tpd]` you get a well known DVI file.

I use the DVI viewer from 4TEX to view the DVI file and I use the freeware ACROBAT viewer to view and print the PDF output

Note: the file `pdftexlib-0.12.zip` unzipped with a few files longer than 8 & 3 characters and I copied these long filenames to shorter ones, keeping the originals in place. I also tweaked the ensuing directory structure, taking out two empty subdirs leading to the filled ones.

Finally, for your interest, in my TEXMF.CNF I have the following lines:

```
% my copy of texmf.cnf
% I saved the original as texmf.ori
% Enable system commands via \write18{...}?
shell_escape = 1

% advised to me by Hans Hagen:
openout_any = r

% Paths

# these kept simple, could maybe me more specific:
TEXMF          = f:/cont-pdf/
TEXMFMAIN      = f:/cont-pdf/
TEXPOOL.PDFTEX = f:/cont-pdf/
TEXPOOL        = f:/cont-pdf/
TEXFORMATS     = f:/cont-pdf/

TEXINPUTS      = .;f:/4texv4/emtex/context/;c:/ourfiles/texinput//

# NB: all FORWARD slashes and "//" ipv 4TEX's "!" voor search
VFFONTS        = f:/mytex/vf/;f:/4texv4/texfiles/VF//
TFMFONTS       = f:/4texv4/emtex/tfm//;=f:/mytex/tfm//

TEXFONTMAPS  = .;f:/cont-pdf//;f:/mytex/psfonts;f:/4texv4/emtex/ps/FONTS//

# T1 fonts, like .afm en .pfb
T1FONTS        = .;f:/cont-pdf//;f:/mytex/psfonts//;

# location of pdftex.cfg,  *.map, *.enc en "PNG images"
TEXPSHEADERS   = .;f:/cont-pdf//;f:/mytex/psfonts//;

AFMFONTS       = .;f:/cont-pdf//;f:/mytex/psfonts//;

MPPOOL         = f:/cont-pdf//
MPMEMS         = f:/cont-pdf//

# to look at later...
MPINPUTS       = .;t:/pragma/metapost//
MFINPUTS.mpost = .;t:/pragma/metapost//

WEB2CDIR    = f:/cont-pdf//
TEXMFCNF    = f:/cont-pdf//
```

```
% TEX settings, maximize for your memory size

main_memory      = 1000000  % 2500000
extra_mem_top    =       0  %  100000
extra_mem_bot    =       0  %  100000
font_mem_size    =  100000  %  500000
font_max         =     500  %     750
hash_extra       =   25000

pool_size        =  400000
string_vacancies =   25000
max_strings      =   50000
pool_free        =  225000

trie_size        =   64000
hyph_size        =    1000
buf_size         =    5000
nest_size        =     750
max_in_open      =      15
param_size       =    1500
save_size        =    8000
stack_size       =    1500

dvi_buf_size     =   16384

error_line       =      79
half_error_line  =      50
```

That's it. Those who are daring, go ahead and try your luck! Meanwhile it can help if you know that a local ConTeXt/PDFtex guru is available and near his phone . . .

Voetnoot recycling

weet iemand hoe ik een eerder gedefinieerde footnote opnieuw kan gebruiken?
ik bedoel dus zoiets als:
hier`\footnote{algemene voetnoot}` en hier`\footnote{algemene voetnoot}`, maar dan maar 1 nummertje en 1 voetnoot onderaan de pagina...
in het geval van de laatste voetnoot ben ik er al uit: `\footnotemark[\value{footnote}]`
maar wat als er nu een voetnoot tussen zit?
hier`\footnote{voetnoot 1}` en hier`\footnote{voetnoot 2}` en hier`\footnotemark[\value{van voetnoot 1 ??}]`
kan dit eigenlijk wel 'netjes'?

Hier een voorbeeldje.
Je kunt ook de 'footnote' counter op en neer schuiven met `\addtocounter`.

```
\newcommand{\footsav}[1]{\xdef#1{\thefootnote}}

text\footnotemark\footsav{\footrefx}
more text\footnotemark\footsav{\footrefy}

again\footnotemark[\footrefx] & and again\footnotemark[\footrefx]
```

```
\footnotetext[\footrefx]{voetnoottext 1}
\footnotetext[\footrefy]{voetnoottext 2}
```

Je kunt helaas niet het normale `\ref` mechanisme gebruiken, omdat het niet ‘expandable’ is.

Ponden en dollars blunderen

Vlak voor definitieve verzending van een fax met financiele afspraken zag ik dat LaTex me bijna een grandioze blunder had laten maken: Je zou toch zeggen dat:

```
\documentclass{article}
\usepackage{times}
\begin{document}
The symbol \pounds{} is used for the english pound.
\end{document}
```

het symbool voor engelse ponden zou laten zien. Niet dus: het blijkt een dollar te worden! Als je het times-package eruit laat gaat het wel goed. Weet iemand waarom dit fout gaat?

Bij mij staat er toch echt een pound symbool. Waarschijnlijk staat er ergens een font setup niet goed bij je.

Dat is, voor mij, erg cryptisch. Hoe controleer ik of een font (het times font blijkbaar) is ge-setup? En hoe set ik het up?

Als je de standaard psnfss files hebt geïnstalleerd en die worden inderdaad gebruikt, en de laatste versies van de vf files zijn er ook dan zou het goed moeten gaan. Ergens zal daar wel iets niet goed staan.

Kijk in je log file. daar moet iets staan als:

```
LaTeX Font Info:  Try loading font information for OT1+ptm on input line 4.

(/import/june/sw/pkg/tex/lib/texmf/tex/latex/psnfss/ot1ptm.fd
File: ot1ptm.fd 1997/02/11 Fontinst v1.6 font definitions for OT1/ptm.
)
LaTeX Font Info:  Font shape `OT1/ptm/m/ui' in size <10> not available
(Font)            Font shape `OT1/ptm/m/it' tried instead on input line 5.
 [1
```

En in ot1ptm.fd:

```
\DeclareFontShape{OT1}{ptm}{m}{it}{
   <-> ptmri7t
}{}
...

\DeclareFontShape{OT1}{ptm}{m}{ui}{<->ssub * ptm/m/it}{}
\DeclareFontShape{OT1}{ptm}{b}{ui}{<->ssub * ptm/b/it}{}
```

Doe een tex testfont vul font `ptmri7t` in en geef `\table\bye`. Bekijk of print de dvi file en zie of er tussen `#` en `%` een pound of een dollar staat.

Je kunt het ook ‘hard’ doen, bv:

```
\def\pound%
  {\bgroup
   \font=cmsi10
   \char36 % of zo
   \egroup}
```

Het probleem zit 'm er in dat de tex fonts niet dezelfde encoding hebben, soms staat in math iets ergens anders.

Ik kan je wel nette macros geven, maar daar heb je niets aan, tenzij je eigen fontswitches inbouwt.

```
\def\dollar%
  {\bgroup
   \ifnum\fam=\itfam
     \sl
   \else\ifnum\fam=\bifam
     \bs
   \fi\fi
   \$%
   \egroup}

\def\sterling%
  {\bgroup
   \ifnum\fam=\bffam
     \bi
   \else\ifnum\fam=\bifam
     \bi
   \else\ifnum\fam=\bsfam
     \bi
   \else
     \it
   \fi\fi\fi
   \$%
   \egroup}

\def\florijn%
  {\bgroup
   \ifnum\fam=\bffam
     \bi
   \else\ifnum\fam=\bifam
     \bi
   \else\ifnum\fam=\bsfam
     \bi
   \else
     \it
   \fi\fi\fi
   f%
   \egroup}
```

Er zijn geruchten dat er versies van `ot1ptm.fd` zijn waarin de ponden niet goed werken. Ik heb de file ook uitgeprobeerd op mijn PC/Windows95 thuis (ik ben langzamerhand aan het omscahkelen van emtex naar miktex) en op miktex gaat het goed. Echter met dviwin stond er in eerste instantie een `$` i.p.v. een pond. Dit bleek te komen doordat dviwin geen pk files voor Times-Roman had, en er dus maar cmr10 voor nam. En dat

gaat fout. In een log file staat dat uiteindelijk dan wel genoemd maart die zie je standaard niet. Heel gevaarlijk dus. Ik heb nu de pk files voor de abdobe fonts van de TeX-live CDROM getrokken en in miktex geinstalleerd, en met yap ziet het er nu goed uit.

De belangrijke regels in je ot1ptm.fd zijn deze:

```
\DeclareFontShape{OT1}{ptm}{m}{ui}{<->ssub * ptm/m/it}{}
\DeclareFontShape{OT1}{ptm}{b}{ui}{<->ssub * ptm/b/it}{}
```

Als die er niet instaan moet je als de wiede weerga een nieuwe set installeren.

EPS file in overhead sheet

Ik moet een template maken voor een overhead sheet. Hierop moet een eps-file komen en daar overheen door de gebruiker van de template te definieren tekst.

```
 +---------------------------------+
 |xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx|
 |xxxxx                           x|
 |xxxxx  Hier moet de tekst dus   x|
 |xxxxx  komen......              x|
 |xxxxx                           x|
 |xxxxx                           x|
 |xxxxx                           x|
 |xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx|
 |xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx|
 +---------------------------------+
```

Kan iemand me zeggen hoe ik dit aanpak. Ik heb al iets geprobeerd met een minipage, maar die lijkt, afhankelijk van de hoeveelheid tekst die er in staat, in hoogte te verschuiven. Ook de EPS file schuift vrolijk mee....

```
\setlength{\unitlength}{1mm} % of zo iets
\begin{picture}(250,180) % breedte, hoogte
  \put(0,0)(\includegraphics{fig.eps}
  \put(20,160){\parbox[t]{10cm}{some text ....}}
\end{picture}
```

Door de [t] optie begint de parbox altijd op dezelfde hoogte. i.p.v. `\parbox` kan ook een minipage, mits ook met [t].

Stippel in tabular?

Ik kom hier niet uit en voor ik de style file in ga duiken: Hoe kan ik in plaats van een `\hline` een stippellijn in een tabular omgeving krijgen?

Gebruik `hvdashln.sty`

Hier is wat grove code die in ieder geval wel een beetje doet wat je wilde:

```
%%BOF dotline.sty
%
% caller syntax as in \cline: \dotline{1-8}
%
% adjust \dotrulewidth as desired.
%
```

```
\newdimen\dotrulewidth
\dotrulewidth = 8\arrayrulewidth

\def\dotline#1{\@dotline#1\@nil}
\def\@dotline#1-#2\@nil{%
  \omit
  \@multicnt#1%
  \advance\@multispan\m@ne
  \ifnum\@multicnt=\@ne\@firstofone{&\omit}\fi
  \@multicnt#2%
  \advance\@multicnt-#1%
  \advance\@multispan\@ne
  \leaders\hbox{%
        \vrule \@height \arrayrulewidth \@width .5\dotrulewidth
        \kern \dotrulewidth
        \vrule \@height \arrayrulewidth \@width .5\dotrulewidth
        }\hfill
  \cr
  \noalign{\vskip-\arrayrulewidth}}

\endinput
%%EOF dotline.sty
```

Symbolen voor natuurlijke getallen

In de standaard emtex versie zijn die, meen ik, niet gedefinieerd. Waar zij ze te vinden?

Ik meen in AmsTeX. Maar ik gebruik zelf voornamelijk zelfgemaakte:

```
\def\RR{I\hspace{-1mm}R}
\def\NN{I\hspace{-1mm}N}
\def\II{I\hspace{-1mm}I}
\def\ZZ{Z\hspace{-1ex}Z}
\usepackage{amssymb}
\newcommand{\C}{\ensuremath{\mathbf{C}}}
\newcommand{\R}{\ensuremath{\mathbf{R}}}
\newcommand{\Q}{\ensuremath{\mathbf{Q}}}
\newcommand{\Z}{\ensuremath{\mathbf{Z}}}
\newcommand{\N}{\ensuremath{\mathbf{N}}}
```

werkt goed bij mij.

Als je de ‘dubbele’ symbolen wilt gebruiken: `\usepackage{amssymb}`
en dan:
`$\mathbb{N}, \mathbb{R}$` etc.

Als je over het pakket `bbm` beschikt kun je ook het volgende doen:

```
\documentclass{article}
 \usepackage{bbm}

 \newcommand{\C}{\ensuremath{\mathbbm{C}}}
 \newcommand{\R}{\ensuremath{\mathbbm{R}}}
 \newcommand{\Q}{\ensuremath{\mathbbm{Q}}}
```

```
\newcommand{\Z}{\ensuremath{\mathbbm{Z}}}
\newcommand{\N}{\ensuremath{\mathbbm{N}}}
```

Je ziet dat mijn antwoord veel op dat van Bas Spitters lijkt. Het verschil is het gebruikte pakket, `bbm` in plaats van `amssymb`, en het daaruit voortvloeiende commando `\mathbbm` in plaats van `\mathbf`. Als alternatief kun je bij de pakketten amsfonts en amssymb ook het commando `\mathbb{N}` enz. gebruiken. Het verschil zit 'm in de plaats waar de extra streep in de N getrokken wordt.

Mocht je ook nog over het pakket `stmaryrd` beschikken dan kun je ook `$\lightning$` gebruiken. Dit bliksemschichtje wordt wel gebruikt als symbool voor TEGENSPRAAK.

Welke maten heb ik?

Hoe kan ik er achter komen wat de textheight en de textwidth van documentclassbook standaard zijn? Staat niet in het L. Lamports LaTeX boek. Ik heb reeds even in bestanden als book.sty enz. rondgesnuisterd. Maar is er geen verbose mode? Of liever is er geen commando voor in je brontekst die waarden voor gespecifieerde variabelen in het log bestand plaatst of zo?

```
\documentclass[12pt,a4paper]{article}
\pagestyle{empty}
\newcommand{\Mrk}{\rule{1pt}{1ex}}
\newcommand{\test}[2]{\string#2&\the#2&#1&\Mrk\rule{#1#2}{1pt}\Mrk\\}
\begin{document}\noindent
\begin{tabular}{@{}lrcl@{}}
name&length&fac&fac x length\\\hline
\test{1}{\evensidemargin}
\test{1}{\footskip}
\test{1}{\headheight}
\test{1}{\headsep}
\test{1}{\hoffset}
\test{1}{\marginparpush}
\test{1}{\marginparsep}
\test{1}{\marginparwidth}
\test{1}{\oddsidemargin}
\test{1}{\parindent}
\test{.3}{\textheight}
\test{.6}{\textwidth}
\test{1}{\topmargin}
\test{1}{\topskip}
\test{1}{\voffset}
\end{tabular}
\end{document}
```

Er is ook een style-file die alle maten voor jou afdrukt en wel `layout.sty`. Dus als volgt:

```
\usepackage{layout}
\begin{document}
\layout
Tekst
\end{document}
```

Utility voor EPS vergroting

Kent iemand een utility om een EPSF plaatje paginavullend uit te vergroten tot A3 en A4

door de hiervoor benodigde PS code eromheen te plaatsen?

ja, TEX.
Ik zou doen:

```
\starttekst

\stelpapierformaatin
  [A3]

\definieeroverlay
  [figuurtje]
  [{\externfiguur
      [koe.eps]
      [breedte=\overlaywidth,hoogte=\overlayheight]}] % isometrisch

\stelachtergrondenin
  [pagina]
  [achtergrond=figuurtje]

\startstandaardopmaak
 % lekker leeg
\stopstandaardopmaak

\stoptekst
```

Format bakken

Een tijdje geleden sprak Alexis Kotte over het maken van een eigen format waar de packages die je vaak gebruik al ingebakken zitten zodat LaTeX iets sneller kan draaien.
Dat zou erg mooi zijn maar met de aanwijzingen die in het bestand `mylatex.ltx` van David Carlisle staan kan ik niet echt overweg.
Wie kan dit nog eens uitleggen?

Dit gaat als volgt:

- Eerst een nieuwe format file aanmaken mbv het volgende commando:

  ```
  initex "&latex" mylatex.ltx file.tex
  ```

  NB. Ik gebruik MiKTeX, en dan gebruik je initex op bovenstaande manier. Een andere TeX-versie heeft hier weer vaak een ander commando voor, bijv. `tex /i` of `tex -i` (geloof ik).
  Het bestand file.tex is hier dus je master-file waarin `\documentclass[]{}` en de `\usepackage[]{}` commando's staan.
- Er is nu een format file aangemaakt (`mylatex.fmt`), deze zet je bij de andere format files of in dezelfde directory als je `file.tex`.
- Je kunt nu je file.tex compileren mbv het volgende commando:

  ```
  tex "&mylatex" file.tex
  ```

  NB. Dit is ook weer voor MiKTeX, ik weet niet of het bij een andere TeX-versie anders werkt.

Voetnoot in titel

Hoe zet ik een voetnoot in een chapter titel? Zowel `\footnote` als `\footnotemark` werken niet.

Gebruik `\protect\footnote`

In principe zou je `\protect\footnote` kunnen gebruiken. Maar je krijgt je voetnoot dan ook in de inhoudsopgave en waarschijnlijk wil je dat niet. Dan kun je de titel zonder de `\footnote` als optioneel argument meegeven (dus tussen []), en dan is de `\protect` overbodig.

Robin Fairbairns heeft een package(je) gemaakt om footnotes is chapter/section titels wat onschadelijker te maken:

`macros/latex/contrib/other/misc/stblftnt.sty`

Footnote in quotation

Ik zou graag in een langere quotation (in de quotation environment) gebruik maken van twee soorten footnotes, die van degene die ik citeer en die moeten dus binnen de quotation, die van mezelf, en die moeten er dus buiten.

De minipage omgeving heeft een apart footnote gebeuren, maar je wilt je kwootjes wqaarschijnlijk wel over pagina's heen laten gaan. In dat geval kan je met een trucje wel de minipage footnotes gebruiken in je kwoot omgeving en ze dan doot een echte minipage laten afdrukken (waar dan verder niks meer instaat. Een beetje smerig misschien en als je wat code uit minipage sloopt en aanpast kun je het nog beter maken:

```
\makeatletter
\newcommand{\qfootnote}[1]{{%
  \def\@mpfn{mpfootnote}\def\thempfn{\thempfootnote}%
  \columnwidth\linewidth\let\@footnotetext\@mpfootnotetext\footnote{#1}}}
\makeatother
\newcommand{\doqfootnotes}{\begin{minipage}{\columnwidth}\end{minipage}}
```

Je gebruikt nu `\qfootnote` voor de footnotes in de kwoot en `\footnote` voor je eigen voetnoten (die dan onder aan de pagina komen).

Als het tijd is om de qfootnotes af te drukken geef je `\doqfootnotes`.

Je kunt de nummering veranderen door `\thempfn` een andere waarde te geven (bijv. `\def\thempfn{\roman{mpfootnote}}` voor Romeinse). Of met `\setcounter` de mpfootnote zetten.

Conversie van / naar LaTeX

The url of this page is `http://www.kfa-juelich.de/isr/1/texconv/texcnven.html`

Although this page resides on the official WWW server of Forschungszentrum Juelich GmbH, it is NOT officially supported by Forschungszentrum Juelich but results from my personal work.

Because this FAQ list grew to a size where it became difficult to manage, I restructured it.

The detailed descriptions were stripped from the main part and put into separate html files, each of them containing the description of only one converter. Additionally, the main list is now divided into two parts:

- Converters from LaTeX to PC textprocessors
- Converters from PC textprocessors to LaTeX

Is er een mogelijkheid om Context op eenvoudige wijze 2 pagina's naast elkaar op een landscape-formaat A4 te laten zetten? Ik bedoel dus eenzelfde uitkomst als psnup -2, maar in mijn geval voor pdf.

Dit kan inderdaad.

Je kiest eerst het goede papierformaat:

```
\stelpapierformaatin[A5][A4]
```

Wellicht moet je de layout nog aanpassen (de standaardinstellingen schalen overigens netjes mee).

De laatste run voeg je toe:

```
\stelarrangerenin[2UP,geroteerd,dubbelzijdig]
```

Let wel, alleen de laatste run!

Voor pdftex output zeg je:

```
\steluitvoerin[pdftex]
```

Handiger is echter het gebuik van texexec:

Je stelt dan alleen het papierformaat in, en tijden het texen de rest:

```
texexec --pdf --output=2up --paper=a5a4 <filenaam>
```

In dat geval wordt eerst een en ander netjes gezet in de standaard arrangering (dit is nodig om alle nummeringen ok te krijgen) en in een laatste run wordt het boekje gemaakt.

Het best is wat te experimenteren met het maps artikel bij de hand, je kunt namelijk ook dingen doen als:

```
\stelpapierformaatin[A6][A4] \stellayoutin[nx=2,ny=2]
```

Dolk in de auteur

Hoe voorkom ik dat de `\thanks` bij een tweede auteur in `\author` een dagger oplevert? Het lijkt alsof de man in kwestie niet meer onder ons is.... Ruud

Het teken wordt bepaald door de footnote teller. Ik denk dat `\stepcounter{footnote}` zou moeten helpen (niet getest).

In a paper with several authors I use multiple `\thanks`. The footnotes are marked by a star, a cross and a double cross. As the second author has not yet deceased i'd like the footnotes to be market by e.g. (a, b, and c). How can i change the standard marking of `\thanks` footnotes.

With thetitlepage option: (but then you wouldn't have had the cross anyway):

```
{
\renewcommand{\thefootnote}{\alph{footnote}}
\maketitle
}
```

without the titlepage option:

```
{\makeatletter\let\@fnsymbol\@alph\makeatother
\maketitle
}
```

fataal erreur 2000, geen millenniumprobleem

On the local W95 machine with 4TEX version 4 on board, we want to view a document

and get ‘fatal error 2000: out of memory’ just before viewing page one.

DVISCR does strange things with Virtual Fonts (PS fonts are handled with VF fonts), i.e. it doesn’t use EMS/XMS for these fonts. So you easily can run out of memory (read DOS memory). There are two possible solutions:

- use DVISCRS: note the S
- Devirtualize the DVI file with DVICOPY

How: 4TeX! i.e. in the main menu use CTRL-D to walk through these options

Verkeerde maten pagina in PDFreader

Als je eenmaal bezig bent volgt er meer. Het valt me ook op dat een blz in acroreader niet het formaat van een landscapeslide heeft. Ga ik via dvips en distiller, dat kost me m’n referenties helaas dan is een sheet wel op het juiste formaat.

In ConTeXt gaat dat als volgt:

```
\stelpapierformaatin
  [S6]  % of een andere S waarde

\stelinteractiein
  [status=start]

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

\stelinteractieschermin
  [breedte=passend,
   hoogte=passend,
   optie=max]
```

En dan ben je pakweg bladvullend bezig.
Ook mogelijk:

```
\stelkleurenin
  [status=start]

\stelachtergrondenin
  [pagina]
  [achtergrond=kleur]
  achtergrondkleur=groen]
```

Welke resolutie nar de drukker?

Als ik een PS file klaarmaak voor printen elders, maakt het dan uit of ik kies voor een 600dpi printer, een 1275 dpi printer of zoeen met ik geloof 2500 dpi? Ik bedoel, levert dat inhoudelijk verschillende files op, of is dat alleen van belang als er bitmapjes in de

file zitten?

Als je dvips gebruikt kan het uitmaken, omdat dvips de resolutie van de `\hrules` en `\vrules` en de plaatsing van letters e.d. ook laat afhangen van de printer resolutie. Aan de andere kant: bij resolutie groter dan 600 doet dat er niet al te veel meer toe, dat verschil ziet het oog toch niet of nauwelijks (gesteld dat er geen bitmaps inzitten, niet al te dunne `\hrules` en `\vrules`, en geen stukken tekst kleiner dan pakweg 6 punts).

Als je iets wilt doen voor printen elders en je weet niet welke printer ze daar hebben is 600dpi best prima. Als je wel weet welke printer ze gebruiken, moet je de precieze resolutie van de printer daar gebruiken (dus niet de 1275 als hun res. 1200 is).

Tip: dvips optie voor verzending naar drukkers

Als je PDF aanlevert die met behulp van dvips en distiller is gegenereerd, let dan even op:

Gebruik de `-j0` optie van dvips, anders kan het zijn dat er characters wegvallen. Iemand bij ons hier had dat probleem met zijn proefschrift. In feite leverde hij postscript in, maar de drukker haalde dat door de distiller omdat zijn printer PDF moest hebben.

En natuurlijk zorg je ervoor dat je Type1 fonts gebruikt (`-Pcmps` of iets soortgelijks).

`-j0` zet de font subsetting van dvips uit. Feitelijk is dat ook wat wij hier doen. We laten dat niet door de distiller uitzoeken omdat dvips veel meer en makkelijker fonts kan vinden dan de distiller (het is nogal annoying dat de distiller niet aan recursief zoeken doet).

Vervolgens laten we dan de subsetting door de distiller doen, zodat (a) de bestandsgrootte klein blijft (b) we de copyrights niet overtreden.

Een andere algemene waarschuwing voor gebruik van dvips: zorg ervoor dat er *nooit* of te nimmer *wat voor pk font dan ook* in de ps file zit als/wanneer je de resulterende PDFs aan wilt passen in Exchange.

Drukkers voegen pdf bestanden nogal eens samen voor het in positie brengen, dus als je meer dan 1 bestand naar de drukker stuurt geldt dit al altijd; maar je kunt met pk fonts ook niet ongestraft 1 pagina overnieuw distilleren na het editen van de tex source om die naderhand in Exchange ‘eroverheen’ te plakken.

Het probleem hier is dat bitmap fonts in dvips altijd PostScript namen krijgen volgens een strict schema: ‘T1’, ‘T2’, ‘T3’, ‘Txxx’, en voor verschillende DVIs kunnen dat volledig verschillende fonts zijn (het geven van namen gebeurt in aanroep-volgorde in de DVI file)

Voorbeeld:

```
(hoofdstuk1.tex) T1 -> cmsy10
                 T2 -> cmmi5

(hoofdstuk2.tex) T1 -> cmmi10
                 T2 -> cmsy7
```

Als dit samengevoegd wordt is er geen enkele voorspelling mogelijk over welke fonts gebruikt gaan worden, dus boek.pdf bevat dan bijvoorbeeld:

```
(boek.pdf)        T1 -> cmmi10
                  T2 -> cmmi5
```

Gevolg: compleet verkeerde tekens in de uitvoer. Feitelijk is dit een bug in Exchange 3, en de situatie is niet altijd zo erg als ik nu voorstel, maar het principiële probleem is aanwezig.

CD-covers maken

Tijdje geleden een cd-writer aangeschaft, nu wil ik dus fraai cd covers en cd backs printen. Ik heb een style cd-cover gevonden die dit kan. Echter, cd-cover.sty print in landscape mode en op mijn HP Deskjet 510 loopt er dan 2 cm van de pagina. Dat is dus een probleem met de linkermarge (want het is landscape, ik verwerk hem dus ook met dvips -t landscape)...

Ik zal F voor zijn met een antwoord hoe dit in ConTeXt gaat; ik heb er een paar dagen terug weer eens wat gemaakt op een color desk jet. In ConTeXt gaat dit als volgt (midden op de pagina, met cutmarks):

```
\stelpapierformaatin[CD][A4]

\stelkleurenin
  [status=aan]

\stellayoutin
  [hoofd=0cm,voet=0cm,kopwit=.5cm,hoogte=passend,
   rugwit=.5cm,marge=0cm,breedte=passend,
   markering=aan,plaats=midden]

% Een figuurtje op de achtergrond:
%
% \stelachtergrondenin
%   [tekst][tekst]
%   [achtergrond=geintje]
%
% \definieeroverlay
%   [geintje]
%   [{\externfiguur[oeps][breedte=\overlaywidth,hoogte=\overlayheight]}]

\starttekst

\startstandaardopmaak
  ......
\stopstandaardopmaak

\stoptekst
```

tekst naast tabel

Kan iemand mij een tip geven hoe ik een tabel naast de tekst kan krijgen. In de tekst maak ik gebruik van de tabulator omgeving.

```
\begin{minipage}[t]{10cm}
De tekst...
\end{minipage}
\begin{tabular}[t]{lcr...}
de tabel
\end{tabular}
```

In dit geval worden ze aan de bovenkant uitgelijnd (vanwege de [t]). Je kunt ook [b] kiezen voor de onderkant. Default is gecentreerd.

Woordafbreking uitzetten?

Het TeXbook is, wat mij betreft, nogal vaag over manieren om het afbreken domweg uit te schakelen. Ik kan (minstens) vier manieren bedenken waarop dat (denk ik) kan:

- kies een `\language` waarvoor je opzettelijk lege patronen hebt geladen bij iniTeX;
- zet `\righthyphenmin` en `\lefthyphenmin` op 100 of zo;
- zet `\hyphenpenalty` (en verwanten) op 10000
- zet `\pretolerance` op 10000

Wat is nou het beste, meest betrouwbare, of aanbevolen manier, en waarom zijn andere minder goed of verkeerd?

`\pretolerance` op 10000
Is onbetrouwbaar. In extreme gevallen kan het zijn dat TeX toch een afbreekroutine probeert te draaien en dan kan het gevolg zijn dat alle “badness” in n regel gepropt wordt. Methodes 1 en 2 verdelen de badness in zo’n geval beter. Voor zulke gevallen kun je beter geven:

```
\language=255 \tolerance=9999 \emergencystretch=10pt
```

Waarbij het eerste natuurlijk ook de oplossing nr. 2/3 mag zijn.

- De eerste optie (een `\language` waarvoor je opzettelijk lege patronen hebt geladen) Heeft als bijverschijnsel dat evt. `\discretionary`, `\hyphenation` en `\-` nog steeds worden uitgevoerd.
- De tweede optie (`\righthyphenmin` en `\lefthyphenmin` op 100) Heeft als bijverschijnsel dat evt. `\hyphenation` niet, maar `\-` en `\discretionary` wel worden uitgevoerd.
- De derde optie (zet `\hyphenpenalty` (en verwanten) op 10000) geeft geen `\hyphenation`, geen `\discretionary`, geen `\-`. Alleen `\hyphenpenalty` is trouwens afdoende.
- De vierde (zet `\pretolerance` op 10000) is een hele slechte, omdat TeX dan ook niet meer probeert mooie regels te maken. Heeft tevens als bijverschijnsel dat evt. `\discretionary`, `\hyphenation` en `\-` nog steeds worden uitgevoerd.

En dan is er ook nog:

`\hyphenchar\the\font = -1`
Wel `\-`, maar je krijgt dan geen hyphen karakter te zien. Geen `\hyphenation`, wel `\discretionary`, met hyphen character.

en je kunt ook nog alle `\lccodes` op 0 zetten
Heeft als bijverschijnsel dat evt. `\hyphenation` niet, maar `\-` en `\discretionary` wel worden uitgevoerd. Heeft als bijverschijnsel dat `\uppercase` en `\lowercase` case niet veranderen.

gedachtestreepjes

Tijdens de NTG-dag in Leuven vond ik een paar prachtboeken op de Addison-Wesley boekentafel, te weten The Non-Designers Type Book, door Robin Williams.

Volgens Williams is het het beste om het gedachtenstreepje—zoals hier dus—zonder spatie, of beter nog met een minieme spatie tussen woorden te plaatsen. Hele spaties levert optisch grotere gaten in de tekst op en is minder mooi.

In De Leestekenwijzer staat het wel met interword spaties als voorbeeld, maar daar zeggen ze er weer bij dat in Engelse teksten vaak een ‘kastlijntje’ wordt gebruikt, een langer gedachtenstreepje dat ook wat dunner oogt.

Wat heeft TeX eigenlijk? Ik vermoed dat wij een kastlijntje hebben omdat TeX uit het Engelse stamt. Williams raadt ook aan om het Engelse gedachtenstreepje iets in te korten, waarmee ze naar onze Europese vastelandstreep neigt.

Als je de terminologie van K.F. Treebus, Tekstwijzer (SDU), hanteert, is:

- \- : een koppelteken [alleen te gebruiken als verbindings- of afbrekingsteken]
- – : half kastlijntje, of kastlijntje op halve lengte [tussen cijfers]
- — : kastlijntje

Ik ben zelf tegen het kastlijntje. Ik vind het te groot. Als je dat gebruikt, moet je geen spaties eromheen doen. En daar ben ik weer tegen, omdat je dan het risico loopt, dat de lezer twee woorden [het woord er voor en het woord er achter] koppelt, die niet gekoppeld moeten worden. Ik gebruik dus als gedachtenstreepje het halve kastlijntje met kleine witruimte eromheen, dus in LaTeX: woord – tweede woord. Dat is ook wat Treebus adviseert en (als ik het goed begrijp) Williams ook.

We hebben `\endash`, een half kastlijntje en `\emdash`, het kastlijntje

Williams raadt kennelijk een driekwart-kastlijntje aan, die bestaan ook ‘in het echt’, en zitten zelfs in Unicode. Je kunt er 1 in TeX maken met:

```
X--\kern -.25em--Y
```

Europees zetwerk doet gewoonlijk:

- europe`\thinspace--\thinspace` dash (frans)
  of
- europe`\thinspace--\kern-.25em--\thinspace` dash (duits)
  of
- europe`\thinspace---\thinspace` dash (engels)

am---dash zonder spatie is typisch amerikaans

Mijn huidige macro voor het gedachtestreepje is, na enige ontwikkelingen en aanpassingen door taco, als volgt:

```
\def\denk{\leavevmode\thinspace%
       \penalty0 \hbox{--\kern -.25em--}%
                    \thinspace\penalty0}
```

Gebruik: `Ik denk\denk wie niet\denk dat het zo goed is.`

Adreslabels op sticker

Bestaat er ook een stylefile waarmee ik adreslabels kan maken voor op zo’n a4-tje met stickertjes?

`label3x8.sty`

TeX op de mac?

Staat er ergens op Internet een beginners manual waarin een eenzame beginnende TeX user met een *Mac* (eenzaam in de betekenis van ‘geen andere TeX / mac user in de

omgeving') kan lezen hoe je begint met TeX op een mac, dwz welke basispakketten (executables, stylefiles, formats) je waar op de mac moet zetten, waar je die kunt vinden op het net, en hoe je die installeert? Met tot besluit een eerste TeX file die kan worden gecompileerd, geviewd en geprint?

Vrijwel alles dat met Mac en TeX te maken heeft kan worden aangetroffen op

`http://www.esm.psu.edu/mac-tex/`

Bij installatie (beschreven in een 'Readme') van b.v. OzTeX -vanaf die site te downloaden- komt ook een uitgebreide user manual beschikbaar.

Tip: Eurofont

Ik heb twee Euro fonts, elk met één karakter erin. Ee is voor proportionele fonts, de ander voor monospaced fonts. Voor zover ik weet zijn deze fonts publiek. Ze zijn, in dee vorm van twee TrueType fonts, te downloaden vanaf de site van HP (`http://www.hp.com`, ergens bij de printer drivers). Ik heb ze omgezet naar Type1 fonts. Zie

`http://www.squirrel.nl/people/jvromans/Euro.html`

fonts: achtergrond

PostScript Fonts op computers?

Taco Hoekwater
taco.hoekwater@wkap.nl

abstract
Dit artikel geeft een korte inleiding in de interne werking van PostScript computer-fonts en hun coderingen. Dit artikel is een aanpassing van een serie slides die gepresenteerd werden op de ntg bijeenkomst van 22 oktober 1998 in Leuven.

keywords
fonts, computers

Inleiding

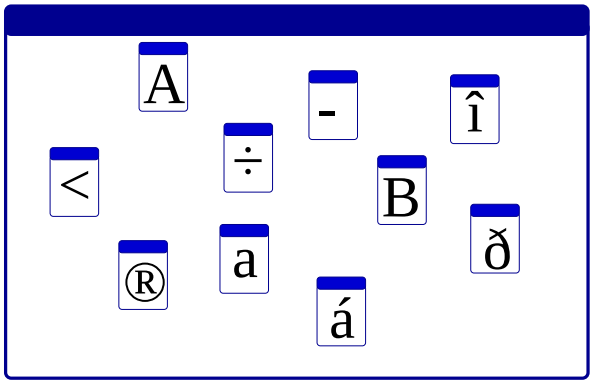
Een computer-font bestaat uit drie logische onderdelen:

- Algemene gegevens van het font
  Hieronder vallen dingen zoals de naam van het font, en de ontwerpgrootte, maar ook zaken zoals kern-paren en hints voor lage-resolutie renderaars.
- Gegevens per karakter
  Karakters hebben meestal zelf ook nog een naam, en die wordt hier bijgehouden. Tevens staan hier dingen zoals de breedte van het karakter.
- Een groep karakters
  Van elk karakter staat hier aangegeven hoe het getekend moet worden. Dit kan per bestandstype verschillen: soms zijn het tekeninstructies, maar soms ook een bitmap plaatje of zelfs alleen een ‘recept’ om dit karakter te maken door bijvoorbeeld twee andere karakters over/naast elkaar te plakken.

Nu we deze algemene eigenschappen van een font gehad hebben, wordt het tijd om eens te gaan kijken hoe een en ander dan wordt opgeslagen op de harde schijf.

PostScript fonts

Een PostScript font onder Windows of Unix bestaat uit twee verschillende bestanden. De algemene gegevens en karakter-gegevens staan in een **afm** of **pfm** bestand. Voor Times-Roman heet dat bestand bijvoorbeeld **tir___.afm** (Bestanden die afkomstig zijn van Adobe Ssytems zijn vaak herkenbaar aan het feit dat Adobe de naam van een bestand aanvult met underscores, zodat het altijd precies acht letters in totaal zijn)



Figuur 1 Schematische opbouw

De karakters zelf staan in een **pfb** of **pfa** bestand. Let op dat zo’n bestand ook *in* een PostScript printer kan staan. In dat geval is het font niet beschikbaar op de harde schijf.

Voor het gebruik van een font binnen TEX is alleen het bestand met de gegevens echt van belang. Het tweede bestand komt pas om de hoek kijken als er ook inderdaad wat moet worden afgedrukt, en valt daarom onder de verantwoording van de DVI driver.

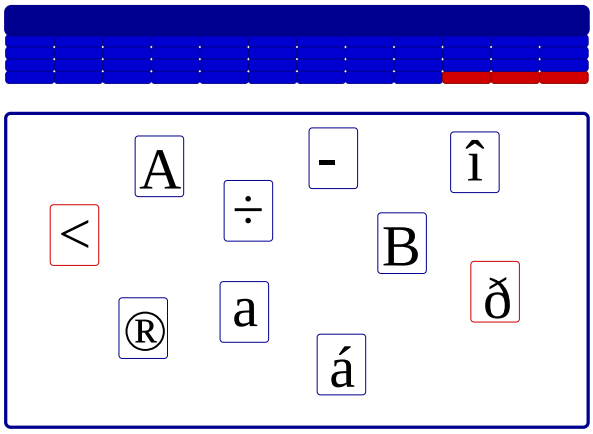
Het is bij het gebruik van een PostScript font handig om rekening te houden met het volgende:

- Er kunnen heel veel tekens in een font zitten.
- Daarvan zijn er maar 256 tegelijkertijd bereikbaar.
- Een **encoding** regelt welke dat zijn.
- Elk font heeft een ingebouwde encoding.
- Maar de kans is groot dat die encoding net niet die tekens bevat die je nodig hebt.
- Daarom is het mogelijk om een font te her-coderen, daarop kom ik later nog terug.

Hoe een driver een font vangt

Even aangenomen dat er een DVI beschikbaar is die PostScript fonts gebruikt en die je graag zou willen printen en/of bekijken. DVI’s zijn onprintbaar, dus hier komt de DVI driver om de hoek kijken. In een DVI staan geen tekens, alleen gegevens.

Kortom: de driver moet op zoek naar het juiste font-bestand. Maar een driver weet niet zeker of een font al dan niet een PostScript font is (die informatie staat niet in de



Figuur 2 Een voorbeeld van een PS font met de ingebouwde encoding

DVI). Er is dus een routine nodig om te beslissen of een font PostScript is, of Metafont source, of ene virtueel font, of een TrueType font, enzovoorts.

Verreweg de belangrijkste driver als we het hebben over PostScript en -fonts is dvips. Dvips gebruikt een opzoek tabel om te kiezen tussen MetaFont (MF) en PostScript (PS) fonts. Die tabel wordt bewaard in een bestand met de naam **psfonts.map**. Deze methode is langzamerhand overgenomen door de meeste andere drivers (zelfs door 'nep'-drivers zoals **pdftex**, ook al heet het bestand dan anders).

De regel om te beslissen of een font al dan niet een PS font is is verbluffend eenvoudig:

- Als het font wordt genoemd in **psfonts.map** is het een PostScript font, in alle andere gevallen niet.

De opzet van psfonts.map

De vorm van het map-bestand is redelijk eenvoudig. Elke regel beschrijft precies n font, en die regel wordt verdeeld in een aantal velden die worden gescheiden door spaties.

Een voorbeeldregel is zoiets als:

```
tnr  TimesNewRoman <tnr.pfb
```

De betekenis van de drie velden hierin is als volgt:

`tnr` Dit is de TEX naam van het font.

`TimesNewRoman`
De PS naam van het font. Die is nodig omdat PostScript interpreters de naam van een font nodig hebben om te kunnen switchen van het ene naar het andere font. De naam kan eventueel ook uit het **.pfb** bestand gehaald worden, maar dvips doet dat niet (nieuwere pdftex's doen dat wel).

`<` betekent: voeg het volgende bestand toe aan de PostScript uitvoer. We zullen later zien waarom het handig is om daar een apart karakter voor te hebben.

`tnr.pfb` Het font zelf. Dit is meestal alleen een bestandsnaam, de driver zoekt gewoonlijk op een andere manier wel uit uit welke directory dat bestand moet komen.

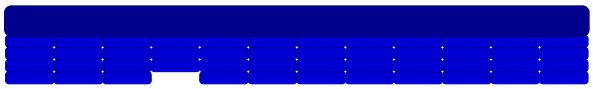
Re-encoding

Het is mogelijk om een PostScript font anders te coderen met behulp van een **encoding**. Daarvoor is wat extra informatie nodig:

- Een bestand met de gewenste encoding, het **.enc** bestand.
- De naam van de encoding.

We nemen weer aan dat zo'n ge-hercodeerd font voor TEX al bestaat: er is namelijk ook een andere TFM voor TEX nodig (doordat sommige tekens nieuw of juist oud zijn kloppen de breedte in het 'standaard gecodeerde' TFM niet meer). Het verschil is grafisch voorgesteld in de twee figuren hieronder:

Figuur 3 Een voorbeeld van een TFM voor PS font met een ingebouwde encoding



Figuur 4 Een voorbeeld van een TFM voor PS font met een aangepaste encoding

Voor **dvips** betekent de her-codering dan dat een extra regel moet worden toegevoegd aan het bestand **psfonts.map**, met bijvoorbeeld de volgende inhoud (de backslashes geven aan dat dit geheel op 1 regel hoort te staan):

```
tnr2 TimesNewRoman                  \
     "TeXnANSIEncoding ReEncodeFont" \
     <texnansi.enc                  \
     <tnr.pfb
```

`tnr2` De TEX naam van het font. Deze naam is nu dus anders

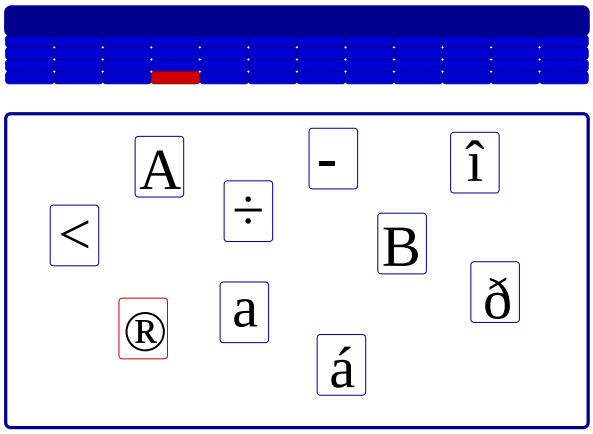
`TimesNewRoman`
De PS naam van het font. Maar deze naam verandert niet door het her-coderen, dus die blijft gewoon staan.

`TeXnANSIEncoding`

De naam van de encoding, zoals die gedefinieerd wordt in het **.enc** bestand.

`texnansi.enc`
Het bestand waar de encoding in staat. Ook hier zou het mogelijk zijn om de naam door dvips uit het **.enc** bestand te halen, maar dit gebeurt niet (pdftex doet dit weer wel).

`tnr.pfb` Het font zelf, wat natuurlijk nog steeds precies hetzelfde bestand is. Immers: de tekens veranderen niet, ze worden alleen anders gerangschikt.



Figuur 5 Een voorbeeld van een PS font met een aangepaste encoding

Virtuele fonts

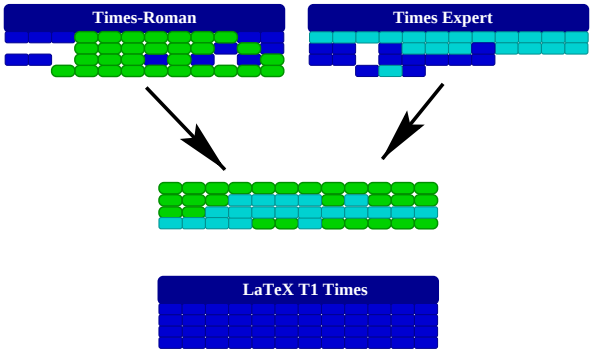
Virtuele fonts zijn weer een verhaal apart, en de bron voor een hoop extra complicaties. Het idee van een virtueel font is relatief eenvoudig: de gegevens die TEX te zien krijgt in n **.tfm**, komen dan eigenlijk uit twee of meer 'echte' fonts. Een speciaal extra bestandje met als extensie **.vf** bevat dan de extra informatie die een driver nodig heeft om uit dat ene samengestelde font de verschillende onderdelen op te vissen.

Een virtueel font gebruikt intern n of meer normale TEX TFM bestanden als bron voor de gegevens. Deze fonts *kunnen* PostScript fonts zijn, maar dat hoeft niet. Deze fonts kunnen ook al ge-hercodeerd zijn, maar dat hoeft ook niet per se. Als laatste is het mogelijk om in een **.vf** bestand een 'recept' te bouwen voor het maken van een karakter (bijvoorbeeld een accent uit n font en een hoofd-karakter uit een ander).

Virtuele fonts worden in deze MAPS bijvoorbeeld gebruikt om de 'mooie' cijfers 12345 uit een ander font te halen. In het 'normale' Times font staan 'saaie' cijfers zoals 12345. Voordeel van deze aanpak als gebruiker is dat ik de vorige regel getypt heb als: `cijfers 12345 uit ... zoals $12345$`; dus zonder dat ik extra commando's in hoefde te typen.

Voor TEX is een virtueel font absoluut niet anders dan een normaal font (wat logisch is, per slot van rekening heeft TEX alleen maar de maten van het doosje om een teken nodig).

Maar dat gaat voor een DVI driver niet op, natuurlijk.



Figuur 6 Een virtueel font

Zoek de verschillen

Een DVI postprocessor zoals **dvips** gebruikt de volgende (alweer verdacht simpele) regel om te beslissen of een font virtueel is en er dus extra werk gedaan moet worden om de onderdelen bij elkaar te zoeken; of dat het een 'normaal' font is

- Als er een bestand met extensie **.vf** bestaat voor een font dat voorkomt in een DVI file *en* het komt niet voor in **psfonts.map**, dan is het een virtueel font.

Dvips gaat dus voor *alle* fonts die in de DVI genoemd worden *eerst* op zoek naar een bijpassende **.vf**. Als zo'n bestand bestaat, dan gaat dvips op zoek in die **.vf** om te ontdekken welke fonts dan wel gebruikt moeten worden. Vervolgens wordt voor de fontnamen die dan gevonden worden weer overnieuw besloten of ze al dan niet virtueel en/of PostScript zijn. Dat gaat zo door tot alle fonts gevonden zijn.

De belangrijkste regel die hieruit volgen is de volgende: Virtuele fonts horen *niet* in **psfonts.map** te staan. Als dat toch gebeurt gaat het gegarandeerd fout.

Mogelijke problemen met fonts

De regels hierboven zorgen ervoor dat vrij makkelijk kunnen controleren wat voor soort font-problemen bij wat voor soort fouten in de TEX-installatie horen.

- Onvindbare bestanden van alle soorten
  Er staat ergens een zoek-pad verkeerd.
  Drivers moeten natuurlijk alle bestanden wel

correct kunnen vinden, gewoonlijk zoeken ze niet de hele harde schijf af (dat zou wel heel erg lang kunnen duren). Dit probleem laat zich oplossen door grondig de documentatie bij de TEX-installatie te lezen en soms ook door bestanden te verplaatsen.

Een smerige truuk die erg vaak werkt: als je weet welk bestand er niet gevonden wordt, kun je dat bestand kopiren naar de directory waarvandaan je de DVI driver aanroept. 10 tegen 1 dat het bestand dan *wel* gevonden wordt.

- Missende **tfm** bestanden
  Dit gebeurt eigenlijk alleen met externe DVIs die je van iemand anders hebt gekregen. Als het toch een bestand van jezelf is, heb je f een paar bestanden gewist sinds de compilatie, f het zoekpad voor de driver staat verkeerd (zoals hierboven), f je installatie is zo ernstig in de war dat alleen een expert of volledig her-installeren je nog kan redden.
- Missende **pfb** bestanden
  Als **dvips** dit tegen je zegt, betekent dat dat het een regel heeft gevonden in **psfonts.map** voor dit font (anders zou het niet weten dat het een PS font was). Er zijn twee nieuwe mogelijkheden waarom deze fout kan ontstaan.

  1) De schuldige kan psfonts.map zelf zijn, bijvoorbeeld omdat er een typefout in staat. Het is zelfs mogelijk dat die regel er helemaal niet in hoort te staan (zoals bij virtuele fonts, of wanneer een MF font toevallig dezelfde naam heeft als een PS font).

  2) Het is een commercieel font wat je niet hebt, maar de TEX distributie had er al wel **.tfm**'s voor meegestuurd. Distributies bevatten vaak als service allerlei bestanden voor fonts die gekocht moeten worden, vooral omdat het maken van de TFM en VF bestanden niet zo heel triviaal is. Deze goedbedoelde toevoeging kan wat misleidend zijn: commercile fonts moeten nu eenmaal gekocht worden.
- Missende **.vf** bestanden
  Deze fout is wat minder snel herkenbaar dan die hierboven. Je merkt dat er wat fout gaat omdat MetaFont wordt aangeroepen dan foutmeldingen begint te geven over niet gevonden **.mf** bestanden.

  Dvips heeft voor zo'n font geen **.vf** gevonden, en ook geen regel in **psfonts.map**, en heeft daarom besloten dat dit dus een Metafont moet zijn.
- Missende regels in **psfonts.map**
  De symptomen van deze fout zijn dezelfde als die van hierboven.

  Voor LATEX gebruikers met een min of meer standaard systeem is meestal te achterhalen welke fout van deze twee het is door te kijken naar de naam van het bestand waar om gevraagd wordt bij Metafont.

  Als die naam eindigt in `8a` of `8r` of begint met `rp` is het probleem waarschijnlijk dat de **psfonts.map** niet goed is.

  Als de naam eindigt op `7t` of `8t` is er meestal een VF bestand zoekgeraakt.
- Verkeerde of missende encodings
  Vanaf hier worden de problemen pas echt lastig. Deze fout en de fouten die hierna komen geven namelijk meestal *helemaal geen* foutmelding, maar zorgen er wel voor dat allerlei accenten verkeerd zijn en/of zelfs voor missende tekens in de uitvoer.

  Het is in het algemeen erg lastig om te ontdekken welke van deze problemen de oorzaak is van de misre.
- Extra regels in **psfonts.map**
  Regels voor virtuele fonts in **psfonts.map** zorgen er soms voor dat een VF onbedoeld verkeerd geïnterpreteerd wordt.
- Teveel TFM bestanden
  Dit is vrij simpel, eigenlijk. Wat er gebeurt is dat TEX en de driver verschillende TFM bestanden vinden, die net niet helemaal precies hetzelfde zijn. De driver roept dan 'checksum error' of iets in die geest.

  Zorg dat je altijd maar n TFM op je harde schijf hebt staan met dezelfde naam.
- Teveel VF bestanden
  Je kunt inderdaad te veel VF bestanden hebben. Als n van de componenten van een virtueel font bedoeld is als 'gewoon' font, maar je hebt per ongeluk een bestand met dezelfde naam maar met extensie **.vf**, dan vindt dvips dat dit *dus* een virtueel font is en begint die **.vf** te lezen.

  Mogelijke fouten kunnen van alles en nog wat zijn, dat hangt immers af van wat er in de **.vf** gedaan wordt. De meestvoorkomende oorzaak van dit probleem: er waren al allerlei bestanden meegeleverd voor het gebruiken van dit font (met de TEX distributie) maar dat wist je niet en daarom heb je een font *nogmaals* geïnstalleerd op een ietwat andere plek.
- Mismatches tussen **tfm** en **vf** bestanden
  Dit gebeurt vooral met DVI's die van iemand anders af komen. In dat geval is er eigenlijk niets aan te doen, behalve dan om een PostScript bestand vragen. Helaas is PostScript een stuk beter portable dan DVI, vooral als er virtuele bestanden gebruikt worden.

En nu?

Samenvattend kan er met fonts heel erg veel mis gaan. Een paar van de dingen hierboven zijn vrij makkelijk herkenbaar, en kunnen ook vrij simpel hersteld worden (bijvoorbeeld als je een PostScript font geïnstalleerd hebt) en je bet

vergeten **psfonts.map** aan te passen.

Problemen met DVI bestanden die van iemand anders komen zijn vrijwel nooit op te lossen. Dan is het vrijwel altijd het best om PostScript of PDF bestanden te vragen in plaats van de DVI.

Andere problemen kunnen heel lastig zijn. Als je vermoedt dat het je eigen schuld is omdat je een stommiteit hebt begaan, is het vaak het best (en in ieder geval het snelst) om het hele TEX systeem te herinstalleren. Als het probleem is ontstaan omdat je probeerde iets zelf uit te breiden, is het waarschijnlijk het best om eerst te herinstalleren en dan nog eens te proberen.

En natuurlijk zijn er voor al dit soort gevallen de guru's van TEX-NL. Blijf niet eeuwig proberen om iets op te lossen wat je eigenlijk niet duidelijk is. Als het over fonts gaat zijn zelfs de simpelste foutmeldingen soms gruwelijk ingewikkeld. Probeer een paar dingen en als het dan nog niet werkt, schroom dan niet op een vraag te posten naar TEX-NL (als je geluk hebt kun je iemand opbellen, dat werkt vaak nog stukken beter).

fonts: tutorial

Maarten Gelderman

How to install a Type1 Font using fontinst

abstract
In this brief tutorial I will describe how a postscript Type 1 font can be made available to TEX using the fontinst-utility (version 1.8). I will not delve into the technical details of this program or the exact functionality of each of the font files used during this process, as those aspects of font-installation are described in other articles in this issue of MAPS. I will not delve into the advanced features of fontinst either, but rather will provide a hands-on tutorial.

keywords
fontinst, Type 1 font

Traditionally typesetting in TEX is done using bitmap fonts – pk-fonts – generated by the METAFONT-program, the most famous or notorious of which is the Computer Modern font, originally developed by Donald E. Knuth for usage with TEX. For typesetting involving advanced mathematics Computern Modern will remain the font of choice, although some commercial alternatives (Times and Lucida from Y&Y (`http://www.yandy.com`) and Helvetiva from Micropress (`http://www.micropress-inc.com`)) are available. For typesetting ordinary text, one may well prefer to use some other, probably commercial font. The majority of those fonts are available in either TrueType or Type 1 (PostScript) format. In this article I will describe how to setup one of the latter fonts, for usage with TEX.[1] Contrary to perceived wisdom, adding an additional PostScript font to your TEX-installation is remarkably uncomplicated.

What do you need?

Of course you need a working TEX-installation. However, that will not suffice. I will also assume that you are using dvips to convert your dvi-files to printable postscript, which in turn supposes that you have a PostScript interpreter, most probably a PostScript-printer, GhostScript or Acrobat Distiller, available. Furthermore, you need the fontinst-utility, which is available from

`ftp://ftp.tex.ac.uk/tex-archive/fonts/utilities/fontinst`

or a similar directory in the CTAN-mirrors `ftp.dante.de` and `ftp.cs.ruu.nl`. I you are using a 'normal' TEX-distribution in combination with a PostScript printer or GhostScript, it is likely that those components have already been installed properly, otherwise you will have to take care of that yourself.

Of course, you also need the font you want to install. A font will typically exist of at least two files. The first eight letters of both file names are identical, the last three, the file names extension, differ. The first file has the file extention `.pfa` or `.pfb`. This is

1. If you want to use TrueType fonts wit TEX, the most easy thing to do seems to use either the program `ttf2pk`, which is available from the `fonts/utilities`-directory of CTAN and from `ftp.freetype.org` or to use `ttftot42`, which is available from `http://ftp.giga.or.at/pub/nih/ttftot42`. Those programs convert your TrueType fonts to ordinary TEX and PostScript fonts, respectively. The latter program 'embeds' the TrueType font in a so-called Type42-font, which can probably be considered a special kind of Type 1 font as far as your TEX implementation is concerned.

the valuable part of the font, the part that contains the letters and that you will probably have to pay for, unless it was supplied as a package deal with some piece of software. The second file you need has the extension `.afm`. It contains information about the main characteristics of the font. If you bought your font, you probably have both files available, if you got the `.pfa` or `.pfb` file as part of some other product, the `.afm` file may be missing. Fortunately, `.afm`-files are distributed for free, and can mosttimes be found at CTAN or the Internet-site of the font-supplier. If you have to obtain your `.afm`-files from one of those sources because it was not supplied with the `.pfa/b` file, make sure that not only the name but also the supplier of the `.afm`-file is identical to that of the `.pfa/b`-file (some popular fonts like Garamond are available from multiple suppliers and the Garamond from Adobe is definitely different from for instance the Monotype variant of this typeface).

A I said in the previous paragraph, you need at least two files to setup the font for usage with TEX. However, the combination of two files, although sufficient technically, can hardly be called a font. The reason is simple. Roughly speaking, in computer parlance a new font is used for each situation in which a single glyph (say an 'a') has a different shape. So your computer needs a font file for the regular a, *the italic a*, **the bold a**, and ***the bold italic a***. Consequently, what is a single font in human perception, is usually made up of at least eight files, two for the regular shaped glyphs, two for the italic, two for the bold and two for the bold italic glyphs. Provided that you have at least one single combination of a `.pfa/b` and `.afm` file, fontinst will happily process them, but you yourself won't probably be satisfied with the result.

Renaming the font-files

Let's suppose that the font files are available. In my case I want to install an Adobe Garamond, which I got together with my scanner. I have got four couples of two files:

- gdbi___.pfb and gdbi___.afm (the bold italic glyphs),
- gdb____.pfb and gdb____.afm (the bold glyphs),
- gdi____.pfb and gdi____.afm (the italic glyphs), and
- gdrg___.pfb and gdrg___.afm (the regular shaped glyphs).

First copy those eight files to a separate directory, so that you can do no harm to the original files.

Now the most difficult part of the installation process has to be carried out. We have to find appropriate names for the font files. A first requirement is that the name you give to your fonts is unique. If you want to mess up your TEX-installation, ignore this requirement and be prepared for some rather unpleasant surprises. A second requirement, which helps you to conform to the first one is that the name you give to your font files is compliant with the font naming scheme developed by Karl Berry, which can be found at

`ftp://ftp.tex.ac.uk/tex-archive/info/fontname`.

This scheme gives an 'algorithm' for generating 'informative' names to fonts within the eight character limit imposed upon us by some old-style operating systems. The name of a font is composed as follows (for a complete and more detailed outline see Karl Berry's original article):

The first letter of the file name is used to indicate which font foundry produced the font we are naming. The Garamond font I am installing here, is produced by Adobe and in Karl Berry's scheme this implies that the first letter of the font name will be a `p`. Had the foundry be Linotype, the first letter would have been an `l`, Bitstream results in a `b`, Monotype in an `m`, etc.

The second and third letter of the name are used to condense the name of the font. In this case I choose `ad` for Adobe Garamond. A rather large, but of course incomplete, list of names is available in the original scheme.

The fourth and possibly fifth letter are used to indicate the shape/variant/width of the font. Again the list of possibilities is huge but when we are installing an ordinary font, the choice is not too complicated. We use the `r` for the regular shape, an `ri` for the italics, a `b` for the bold shape and `bi` for the bold italics.

The next two letters are used to indicate the 'font encoding'. The `pfa/b` files contain a huge number of glyphs. In principle they may be arranged in any random order. Let's for the sake of simplicity assume that the files only contain the letters a to z. One font designer may choose to make the a the first letter in his font, the b the second etc. Another may start with the z and count down. The number of possibilities is sheer endless. Without knowledge of the way the glyphs are arranged (the encoding), the font files are virtually useless. The most easy way to find out how the glyphs are arranged is to examine the contents of the `.afm`-files (`.afm`-files are regular ASCII-files, you can load them directly into your favorite editor). Those files contain information similar to that listed below:

```
StartFontMetrics 2.0
Comment  Copyright (c) 1989 Adobe Systems Incorporated.
Comment Creation Date:Wed Jul 12 19:25:18 PDT 1989
FontName AGaramond-Regular
EncodingScheme AdobeStandardEncoding
FullName Adobe Garamond Regular
FamilyName AdobeGaramond
Weight Regular
```

The information we are looking for is 'EncodingScheme AdobeStandardEncoding'. According to Karl Berry's scheme we now have to use `8a` as the next two letters of our file name. Other more or less frequently occurring coding schemes are Adobe Expert (`8x`), TeXBase1 (`8r`), and TeXnANSI (`8y`).

Optional last letter The optional last letter is not needed in this example, but can be used to indicate the width of a font. Some fonts have a narrow, a condensed, a wide or an extended variant in which case we have to add `n`, `c`, `w`, and `x` respectively to the file name.

Now we are ready to rename the font files into `padr8a`, `padb8a`, `padri8a`, and `padbi8a` for the regular, bold, italic and bold italic shape respectively. Although strictly speaking it is only necessary to rename the `.afm`-files, I personally prefer to rename the `pfa/b` files as well in order to retain their status as matched couples. Use the `rename` or `mv` command or whatever functionality you operating system provides to give the (copies of the) font files the appropriate names.

Running fontinst

Now we have to prepare a TEX-file which will be used by fontinst to process our font files. The contents of this file is rather simple:

```
\input fontinst.sty
\latinfamily{pad}{}
\bye
```

The first line asks TEX to input the fontinst-utility and the next line tells fontinst to process the fontfiles. Note that the three letters `pad` are the first three letters of the name of our

fontfiles and probably will be different if you are processing your own files. Fontinst automatically takes care of the different shapes and encodings of the files to be input, with two exceptions: (1) if you want to install a family using expertfonts you have to add the letter `x` and (2) if you want to install a family using expert fonts with oldstyle digits you have to add the letter `j`. In the two latter cases the command to use will be `\latinfamily{padx}{}` and `\latinfamily{padj}{}`, respectively. Run the file through TEX, look at your terminal and wait (possibly for quite a long time).

TEX will generate an extraordinary amount of messages on your screen. Simply ignore them. The fontinst utility is trying to install almost any imaginable glyph and any imaginable variant of the font at hand. A lot of those glyphs and variants will obviously not be available to you and so fontinst complains and in some cases even finds a solution for lacking glyphs and variants: lacking ligatures will be replaced by their composing characters and slanted and fake small caps variants of a font will automatically be generated if necessary.

Usually TEX will finish without any fatal error messages and a directory listing will show you that some new files have been generated. However, ignoring typos, two situations may occur in which this is not the case. Firstly, TEX may complain about the definition of the `\bye`-command. In this case you are using an old version of fontinst. Upgrade to a newer one. Second, TEX may complain about a lack of memory. Upgrade to a larger TEX or adapt the configuration of the TEX-version you are using.[2]

Processing the output of fontinst

Part of the output generated by fontinst needs an additional conversion before it can be used by TEX. This conversion is done by issuing the following two commands from within your command-shell (Dos-syntax on the left, Unix (bash) on the right):

```
for %f in (*.pl) do pltotf %f          for f in *.vpl; do vptovf $f;done
for %f in (*.vpl) do vptovf %f         for f in *.pl; do pltotf $f;done
```

Probably some messages about rounding errors will appear on your screen, ignore them. Just check whether some new `.tfm` and `.vf` files have been generated, which will usually be the case.[3] Again a couple of things may go wrong, in particular on Dos and Windows computers. If either pltotf or vptotf is a batchfile add `call` between do and the program name. A second possible risk is that both programs are somewhat oldfashioned and don't like the syntax used above. As you might have grasped pltotf reads a `.pl` file and converts it to a `.tfm` file. Some older pltotf's require you to enter the full name of both the `.pl` file to be read and the `.tfm` file to be generated. Similarly vptovf reads a `.vpl`-file and converts it to two other files a `.vf` and a `.tfm` file. Some oldfashioned programs, again, require you to explicitly specify the names of all files to be read and generated.

Putting everything in place

Almost all required files have been generated by now. The main thing to be done is place them in the correct directory (when you are using a more advanced operating system like Unix, you will probably need system administrator permissions to complete this final stage). Before moving the newly generated files, let's make sure that we do not generate conflicts by using duplicate filenames. Just execute the following commands (I assume

2. How to do this is largely dependent on your TEX-implementation. If you do not want to upgrade the most easy solution may be to run the fontinst-job on someone elses computer and copy the output afterwards.
3. To be more accurate, for each of the four couples of files used in this case five `.tfm`-files will be generated and three `.vf`-files, furthermore some files for fake fonts will be made.

that you use the TEX Directory Structure; TDS, replace texmf by the appropriate directory on your machine):

```
for %f in (*.tfm) do dir texmf\%f/s          for f in *.tfm; do find /texmf -name $f;done
for %f in (*.vf) do dir texmf\%f/s           for f in *.vf; do find /texmf -name $f;done
for %f in (*.fd) do dir texmf\%f/s           for f in *.fd; do find /texmf -name $f;done
```

You have to move all .tfm-files to a subdirectory of texmf/fonts/tfm;[4] the .vf-files go to a subdirectory of texmf/fonts/vf and the .fd files to a subdirectory of texmf/tex/latex. The .pfb-files also have to be placed in an appropriate directory. Although not strictly required it might be prudent to put them into a subdirectory of texmf/fonts/type1. After moving all those files, you may have to rebuild your hash-tables by typing mktexlsr (just try it, if it is not necessary it will not harm either).

Finally you have to inform dvips about your intent to use your new font by editing psfonts.map (which is probably located in texmf/dvips/base. Add the following lines:

```
padr8r  AGaramond-Regular    "TeXBase1Encoding ReEncodeFont " <8r.enc \
                                        <padr8a.pfb
padri8r AGaramond-Italic     "TeXBase1Encoding ReEncodeFont " <8r.enc \
                                        <padri8a.pfb
padb8r  AGaramond-Bold       "TeXBase1Encoding ReEncodeFont " <8r.enc \
                                        <padb8a.pfb
padbi8r AGaramond-BoldItalic "TeXBase1Encoding ReEncodeFont " <8r.enc \
                                        <padbi8a.pfb
padro8r AGaramond-Regular    "0.167 SlantFont TexBase1Encoding ReEncod\
                                        eFont " <8r.enc <padr8a.pfb
padbo8r AGaramond-Bold       "0.167 SlantFont TeXBase1Encoding ReEncod\
                                        eFont " <8r.enc <padb8a.pfb
```

The preceding lines have been constructed as follows. The first column contains the name of the font according to the Karl Berry scheme, but this time in TexBase1Encoding (8r). The next column contains the name of the font. This name should be copied verbatim from the fontname contained in the .afm file (see the listing on page 39). The next column contains some PostScript commands that convert the font to the desired encoding. The final two columns are used to tell dvips that it should add two files to all output it generates containing the font concerned: the encoding file and the file with the font itself.

The last two lines to be added to dvips are somewhat different from the other four. Most PostScript fonts do not come with a slanted version. Those last two lines are used to generate one. The first column contains the Karl Berry name of the slanted font, the next column the fontname of the font that will be used to generate the slanted font (this same font is added to the PostScript file in the final column). The column with the PostScript commands now contains the additional text '0.167 SlantFont', that generates a slanted version on the fly.

Using the font

Now you are ready to use the font. Just add the command

```
\renewcommand{\rmdefault}{pad}
```

to the preamble of your LATEX document and your newly installed font will be used to typeset it. Probably you will want to take some additional measures in order to ensure that your

4. The exact choice of a subdirectory does not matter, however, you may want to give it a name that makes some sense.

documents appearance will be acceptable. The Garamond font, for instance, looks absolutely ugly when used with the standard LaTeX-classes. Using the classes of the American Mathematical Society (e.g., `\usepackage{amsart}`) already is a considerable improvement. If you use a typewriter font regularly, you may want to switch to Courier instead of the regular Computer Modern typewriter, which is way too black. However, Courier is too large, so you will have to scale it. To accomplish this, I made a file `ot1zcr.fd` by renaming and modifying the `ot1pcr.fd`. By adding `\renewcommand{\ttdefault}{zcr}` to my preamble I obtain the scaled typewriter font instead of the regular one. The interpretation of the stuff in the font definition file underneath is left as an exercise to the reader (the main feature of which is a scalefactor of 89%, further information can be found in Siep Kroonenberg's article elsewhere in this issue).

```
%Filename: ot1zcr.fd
%Created by: Maarten from ot1pcr.fd

%THIS FILE SHOULD BE PUT IN A TEX INPUTS DIRECTORY

\ProvidesFile{ot1zcr.fd}
   [1997/09/30 Fontinst v1.6 font definitions for OT1/zcr.]
   % Modified by Maarten

\DeclareFontFamily{OT1}{zcr}{}

\DeclareFontShape{OT1}{zcr}{b}{n}{ <-> s * [0.89] pcrb7t }{}

\DeclareFontShape{OT1}{zcr}{b}{sc}{ <-> s * [0.89] pcrbc7t }{}

\DeclareFontShape{OT1}{zcr}{b}{sl}{ <-> s * [0.89] pcrbo7t }{}

\DeclareFontShape{OT1}{zcr}{m}{n}{ <-> s * [0.89] pcrr7t }{}

\DeclareFontShape{OT1}{zcr}{m}{sc}{ <-> s * [0.89] pcrrc7t }{}

\DeclareFontShape{OT1}{zcr}{m}{sl}{ <-> s * [0.89] pcrro7t }{}

\DeclareFontShape{OT1}{zcr}{bx}{n}{<->ssub * zcr/b/n}{}
\DeclareFontShape{OT1}{zcr}{bx}{sc}{<->ssub * zcr/b/sc}{}
\DeclareFontShape{OT1}{zcr}{bx}{sl}{<->ssub * zcr/b/sl}{}
\DeclareFontShape{OT1}{zcr}{b}{it}{<->ssub * zcr/b/sl}{}
\DeclareFontShape{OT1}{zcr}{bx}{it}{<->ssub * zcr/b/it}{}
\DeclareFontShape{OT1}{zcr}{l}{n}{<->ssub * zcr/m/n}{}
\DeclareFontShape{OT1}{zcr}{l}{sc}{<->ssub * zcr/m/sc}{}
\DeclareFontShape{OT1}{zcr}{l}{sl}{<->ssub * zcr/m/sl}{}
\DeclareFontShape{OT1}{zcr}{m}{it}{<->ssub * zcr/m/sl}{}
\DeclareFontShape{OT1}{zcr}{l}{it}{<->ssub * zcr/m/it}{}

\DeclareFontShape{OT1}{zcr}{m}{ui}{<->ssub * zcr/m/it}{}
\DeclareFontShape{OT1}{zcr}{b}{ui}{<->ssub * zcr/b/it}{}
\endinput
```

fonts and scripting

Installing PostScript Fonts Under Unix/Linux

Roland Kwee
email:
rolandkwee@acm.org

abstract
This article discusses practical issues for installing Adobe PostScript fonts for TEX running on Unix/Linux. It also presents a script for automating the installation in order to install a package with over 200 fonts. As a useful side effect, the script creates a font data base and a font catalog to help the buyer/user of the font collection.

keywords
fonts, PostScript, linux, Adobe, dvips, ghostscript.

Introduction

At the recent NTG meeting in Leuven, Belgium, I got enthousiastic about fonts, and even bought my first book on typography. After years of using TEX and Latex, I had become familiar with many text processing and formatting issues, and most of you will agree that this is a big field. But after reading the typography book I realized that text formatting is only one part of the job, and that using fonts in an effective manner is the other half. This article addresses people who are convinced that using fonts is a very important issue when creating documents. If you are not convinced of this, I recommend to study one of the introductory books on typography before going into the remainder of this article, because using fonts in the "wrong" way would probably not improve your documents.

In order to use fonts, one first needs to obtain some. The computer modern fonts and the 35 standard PostScript fonts are just not enough for ambitious use of fonts. At the NTG meeting I thought differently, but after reading on typography I know better. I happened to get the Adobe Type–On–Call CDROM from a colleague who once purchased some Adobe fonts. It seems that font vendors distribute CDROMs at no or little charge, because they charge you later for the key codes to unlock the fonts that you want to buy, usually about f 60 per type face (a font usually contains several type faces, e.g., roman, italic, bold). However, upon registering the Adobe Type–On–Call CDROM, you get to choose two free fonts, and in my case, Adobe threw in another free font. The latter was package 950–00, containing over two hundred fonts.

In order to use the obtained fonts, one needs to install those. The vendor usually provides you with an install program for DOS/Windows, or Mac, but not for Linux. Font installation, whether PK or PostScript format, is never simple. Installing over two hundred fonts needs to be automated. This article presents an overview of the issues around PostScript font installation, and a way to automate this.

In order to use the installed fonts, one really needs a font catalog, showing samples of each installed font, and linking those to the font codes used in TEX documents. The scripts presented in this article therefore also create a font catalog as part of the installation procedure. The catalog is also a test of the installed fonts.

This article assumes some familiarity with dvips and ghostscript/ghostview, because those are the programs I use to print TEX documents. The reader is also assumed to understand common shell commands, or to be able to look these up in the man pages.

Font basics

First I will give you an overview of the installation procedure of Adobe fonts. In the following section I'll describe ways to automate the installation.

The description is for Type 1 Postscript fonts. To be honest, I haven't taken the trouble yet to find out about Type 2 fonts.

Font installation under windows The installation of a purchased font must begin under windows, because only then is it possible to use the key code faxed to you by Adobe to unlock a font.

From the CDROM, the following files are supplied for a font:

- `font____.pfb` (postscript font binary) contains the characters, this file must be unlocked with a key to be ordered from Adobe.
- `font____.afm` (adobe font metrics) contains metrics. It is normally not installed under Windows. To get this file, check the appropriate box in the Adobe Installer. It can also be copied from the CDROM, but note that the *hidden* attribute is set.
- `font____.pfm` (printer font metrics) contains metrics for windows, this file is extracted from the afm file during installation.
- `font____.inf` is for windows, this file is also extracted from the afm file during installation.

The Adobe file names are eight plus three characters, and underscores are added for font names shorter than eight characters.

After you installed the fonts under windows, you can immediately start using those, e.g., with notepad. By browsing the list of fonts from the notepad menu, you'll not be able to see whether a name relates to a Windows standard font, or a font purchased from Adobe. Because a font is really one typeface, some fonts must be selected with the bold and/or italic buttons. Under windows, you'll soon miss a home made font data base and font catalog.

If you install more than one font package, you should create a directory for each package and specify those in the Adobe Installer. After all, if you obtained the fonts legally, you want to be able to keep track of the origin of each and every font file in an easy way.

NFSS font name scheme Postscript fonts are sold by various suppliers: Adobe, Bitstream, Linotype, etc. Some fonts have more than one supplier, and those may be identical fonts or variants. This gave rise to all kinds of file name conflicts and other chaos. Now we use the NFSS: New Font Selection Scheme. In this scheme, many fonts of most suppliers have gotten new file names, see the file adobe.map by Karl Berry, available on CTAN. For fonts you may have obtained not listed in that file, you should choose file names that have not yet been assigned.

The NFSS font names include codes for the supplier, encoding, weight, etc. For details, refer to *The Latex Graphics Companion* or other excellent books.

All this helps when you want to see the correct fonts used for your TEX files when processed at other sites.

NFSS encoding The pfb font file defines a set of characters (glyphs), identified by names, such as "A", "A-acute" (Á), "A-ring" (Å), etc. The pfb font file also defines a mapping of glyph names to numbers. In a PostScript document all characters are referenced by these numbers.

In practice, this mapping often does not include all available characters, and sometimes the ordering of the mapping is not convenient. Therefore NFSS has defined some new

encodings, suitable for TEX usage. The program *afm2tfm* has a feature to rearrange the encoding.

Font installation for TeX on Linux To install a postscript font for TEX, several steps are needed. It is worthwhile to try to install a single font file *by hand* just to get familiar with the issues involved.

- Install the font under windows, as described.
- Determine the adobe file name (with the underscores), the official font name (see the FontName header line in the afm file), the desired TEX font name (e.g.: the NFSS FontName naming scheme), and the best encoding: 8r for text fonts, 7a for symbol fonts.
- TeX needs a tfm file (tex font metrics), which can be obtained from the afm file with the program afm2tfm which is commonly installed with TEXThe typical location for the file is texmf/fonts/tfm/adobe/..../font.tfm. Don't forget to run MakeTeXls–R.
- dvips needs a new entry in its psfonts.map file to know that the font is a postscript font, and to translate between the font name in the TEX system and its postscript name. Typical location is texmf/dvips/psfonts.map.
- The printer, in my case the ghostscript interpreter, needs the pfb file. Ghostscript needs an entry in the file Fontmap to associate the pfb file with the postscript font name that dvips writes in the postscript output. The typical directory for file Fontmap is `/usr/lib/ghostscript-4.03/fonts/`.
- In your TEX document, use the commands `\font\f=pabc at 10pt \f` to typeset text in the new font, assuming that you installed the new font under the name pabc.

Note that ghostscript does not complain if it cannot find a font. It just substitutes another font without warnings. If you are not sure which file is used for a particular font, just use ghostscript without the user–friendly shell: `$ gs -sDEVICE=x11 file.ps` This will show some information. Leave the gs shell with `exit` .

Font installation procedure

To install a large number of files, a script can be written that performs all necessary steps on all font files. However, it turned out that a single script will be complex. Many fonts have something special, e.g., about the file name or encoding, and the handling of special cases makes a script extra complicated. A complicated script takes more effort to debug.

A better approach seemed to split the installation in several steps. A first script builts a kind of data base text file with a line for each font and columns for each attribute. Then, after verifying that all attributes are correct, a second script installs the files. Both scripts can be quite simple and straightforward, with a high degree of trust that if one font is installed correctly, all fonts will be installed correctly.

The font data base text file is not only used as an intermediate step for installation, but also after installation as an aid in using the fonts. A third script produces a TEX file that prints samples of each font, i.e., a font catalog.

Creating the font data base

To keep things organized, I asked the Adobe font install program to install package 950–00 in directory `C:\PSFONTS\950-00\` . This is where the pfb files are installed. The afm files

are installed in directory `C:\PSFONTS\950-00\AFM\` . On my computer, these directories are accessible under linux:

```
$ ls /dosc/psfonts/950-00/*pfb | wc
```

$\Longrightarrow$ 227 font files

To obtain the Fontname font names from the file adobe.map with grep, this file must be sorted on the second field. Otherwise, the first line of a `grep` on Garamond-BoldCondensed will give the entry for Garamond-BoldCondensedItalic instead. To sort the file, remove the comments, run this shell command:

```
$ sort -b -k 2 adobe.map > adobe.map.sorted
```

and reinstall the comments.

The following script reads all file names, processes those to extract the base file name etc, and to look up the standard font name and encoding. I present the file with all comment lines, in the hope that these help in understanding the script. I haven't looked very hard for a web version for shell scripts, but if I had it available, I'd written this entire article in the script.

```
#!/bin/sh
# fontdb1.rc     Roland Kwee    30 Dec 1998
# Purpose: build a font data base text file for installing PS fonts.
# Needed files: adobe.map must be in the current dir.
# Argument(s): $1 = package number, e.g., 950-00
# Usage: chmod +x fontdb1.rc;
#        rm -f fontdb1.txt; ./fontdb1.rc 950-00 > fontdb1.txt

package=$1 # Adobe package code, e.g., 950-00

# This function will be run on each pfb file name.
# Argument(s): $1 = pfb file name with or without path.
func(){
  # Make several versions of the file name.
  fname=`basename $1` # remove path name
  fontname=`expr $fname : "\([^_]+\)_*\.pfb"` # e.g., ovbk
  fontn___=`expr $fname : "\([^\.]+\).pfb"`   # e.g., ovbk____
  # Find official font name from the afm file.
  offname1=`grep FontName $sourceafm/$fontn___.afm | tr -d '\015'`
  offname=`expr "$offname1" : "FontName\ \(.+\)$"`
  # Find name in FontName scheme.
  fnname1=`grep $offname adobe.map`
  fnname=`expr "$fnname1" : "\(^.+\)\ .*$"`
  # Not all fonts will be in adobe.map, so mark fonts not found.
  # But if it is found, extract the encoding.
  if [ -z "$fnname" ]
  then
    fnname="!!!!!!!!"
    encoding=""
  else
    # Note that 0 and 2 do appear in font abbreviations,
    # but not in encodings.
    encoding=`expr "$fnname" : "[a-z0-3]+\([4-9][a-z]\).*$"`
  fi
  # Some fonts require no reencoding by afm2tfm.
  if [ -z "$encoding" ]
  then
```

```
    encoding="xx"
  fi
  # Output all attributes thus found in a data base text file.
  # Each line starts with a function name to enable further processing.
  # Use tabs to separate variable-width columns in a nice way.
  echo -e \
   "func  $package  $fontn___  $fontname\t$fnname\t$encoding $offname"
}

# e.g., Officina Book font of package 950-00 is in files:
#   $sourcedos/950-00/ovbk____.pfb, $sourcedos/950-00/afm/ovbk____.afm
sourcedos=/dosc/psfonts
sourceafm=$sourcedos/$package/afm

# Run the above function on each file in the package directory.
for f in $sourcedos/$package/*.pfb
do
  func $f
done
```

After running this script, and saving the output in the font data base text file fontdb1.txt, you have to inspect the data base file to resolve fonts that could not be found in adobe.map. I did that by extending adobe.map with a new file adobe-ext.map as shown below, and running the above script again.

Find all missing names:

```
$ grep \!  fontdb1.txt > adobe-ext.map
```

Next, type new FontName names in file adobe-ext.map. For compatibility, do not re-use abbreviations that are already defined for other fonts, e.g., ov for Octavian. In other cases, the Adobe file name is a good starting point for abbreviations. All names begin with p (supplier code for Adobe). All text file names end with 8r (TEXbase1 encoding), all other fonts (i.e., symbol fonts) with 7a (alternate characters only). For Adobe package 950-00, my file `adobe-ext.map` got entries on the basis of the following new 2-character Fontname abbreviations (Adobe file names in parentheses):

- ir Isadora (ir)
- of OfficinaSerif (ov)
- ow OfficinaSans (ow)
- th Tiepolo (tp)
- rf Rosewood-Fill (uxfi)
- rr Rosewood-Regular (uxrg)
- wm Myriad-Sketch (wmske)
- wq Quake (wq)
- wy Mythos (wy)

After rerunning the script, you must again check very carefully the resulting data base. Especially the encodings are prone to errors, e.g., if the font name has digits. Correct those errors and store the file as fontdb2.txt.

ZapfDingbats appears without encoding, which is now indicated with xx. The program afm2tfm should then not use any encoding.

Check for uniqueness of the font names (5th column) as found in adobe.map, by running the following shell command:

```
$ sort -b +4 fontdb1.txt | uniq -d -4 -w 9
```

which will print only duplicate entries.

Note that it is much more fun to correct entries in the font data base text file than to correct fonts that are installed incorrectly. Programmers and engineers would say that it is cheaper to correct errors earlier in the development process.

Now we add the following shell commands to install the various files. I really had one file containing the script code for all stages of the installation, but here I show only the code relevant for the installation of the files according to the font data base. The preamble and postamble stuff is something that is not really used in this stage, but is meant for producing the font catalog later on.

```
#!/bin/sh
# install.rc            Roland Kwee     30 Dec 1998
# Shell commands to install font files in TeX
# and to create font maps for dvips and ghostscript.
# Usage: create a directory for the new package in the ghostscript dir
#        and in the tfm dir tree, then run this script.

# Stop shell script upon an error.
set -e

func_install(){
  package=$1
  adobename=$2
  fontname=$3
  fnname=$4 # name according to Fontname scheme
  encoding=$5
  offname=$6 # official postscript name
  echo Installing: $offname
  # Create and install tfm file and add entry to map file.
  afmdir=/dosc/psfonts/$package/afm
  tfmdir=/bigdisk1/usr/local/share/texmf/fonts/tfm/adobe/$package
  if [ ! -d $tfmdir ]
  then
    echo Need to do: mkdir $tfmdir
    exit 1
  fi
  dvipsdir=/bigdisk1/usr/local/share/texmf/dvips/base
  # Note: there is no encoding file 7a.enc in dvips, so ...
  if [ "$encoding" = "8r" ]
  then # with encoding
    (cd $dvipsdir; afm2tfm $afmdir/$adobename.afm -T $encoding.enc \
                   $tfmdir/$fnname.tfm >> $package.map)
  else # without encoding
    (cd $dvipsdir; afm2tfm $afmdir/$adobename.afm \
                   $tfmdir/$fnname.tfm >> $package.map)
  fi
  # Copy afm and pfb files to the ghostscript directory.
  # Create a map file for ghostscript.
  pfbdir=/dosc/psfonts/$package
  gsdir=/usr/lib/ghostscript-4.03/fonts
```

```
  if [ ! -d $gsdir/$package ]
  then
    echo Need to do: mkdir $gsdir/$package
    exit 1
  fi
  cp $pfbdir/$adobename.pfb $gsdir/$package/$fontname.pfb
  cp $afmdir/$adobename.afm $gsdir/$package/$fontname.afm
  echo -e "/$offname \t($package/$fontname.pfb) \t;" \
    >> $gsdir/$package/Fontmap
}
# Make it easy to switch between scripts to run on the font data base.
func_preamble(){
}
func(){
  func_install $*
  #func_catalog $*
}
func_postamble(){
}
func_preamble
# Now follow the font data base entries, like:
#func  950-00  agd_____  agd     pagd8r      8r AvantGarde-Demi
#func  950-00  agdo____  agdo    pagdo8r     8r AvantGarde-DemiOblique
# etc.
func_postamble
```

Save the new file with the above script and the font data base entries from fontdb.txt under the name install.rc, and run it as superuser, but don't forget to create the package directories for tfm and pfb files.

If, for some reason, you need to rerun the script, make sure to first remove the font maps generated with the old run.

As a final step, paste the new font map files for dvips and ghostscript to the original ones. Now the files should be installed and ready to use.

Common installation problems

A problem can arise from the default dvips font map file psfonts.map. Many more than the 35 standard postscript fonts are listed. Some may overlap with the added fonts, but specify a font file that is not available. This may lead to most new fonts to become unavailable, perhaps by triggering bugs in dvips. A rigorous cure is to delete all superfluous font names from psfonts.map, before adding the new font names. It was a nice idea to include many fashionable fonts in the psfonts.map file, but in my opinion, the extra lines are just garbage. It is just as easy to update psfonts.map each time you obtain and install new fonts.

A similar problem may arise with the ghostscript Fontmap file. E.g., ZapfDingbats is a standard font and is standard installed. When installing Adobe package 950-00 in a separate font directory, a second ZapfDingbats file is installed. These should be identical, but one of the fonts may never be used. If both font files must be available, it may be necessary to do some renaming in Fontmap.

Font Catalog

The following script generates TeX code for a font catalog. It includes the calls to include a file with TEX macros, and to end a TEX session.

```
#!/bin/sh

# Function to create an entry in a TeX font catalog.
func_catalog(){
  package=$1
  adobename=$2
  fontname=$3
  fnname=$4 # name according to Fontname scheme
  encoding=$5
  offname=$6 # official postscript name
  echo \bs entry{$offname}{$fnname}{$package}{$fontname}
}
# Make it easy to switch between scripts to run on the font data base.
func_preamble(){
  echo %this file is generated by script fontdb.rc.
  echo \bs input catalog.mac
}
func(){
  #func_install $*
  func_catalog $*
}
func_postamble(){
  echo \bs bye
}
func_preamble
# Now follow the font data base entries, like:
#func  950-00  agd_____  agd     pagd8r      8r AvantGarde-Demi
#func  950-00  agdo____  agdo    pagdo8r     8r AvantGarde-DemiOblique
# etc.
func_postamble
```

The following file contains TeX code to typeset the font catalog. Of course, as a dedicated TEX user, you will want to improve this to suit yourself.

```
% catalog.mac     Roland Kwee    1 Jan 1999
% TeX macros for fontdb.rc.

\font\c=ptmr at 10 pt \c

\def\entry#1#2#3#4{%#1=official ps name, #2=tex font name,
                   %#3=adobe package code, #4=adobe file name
  {\font\f=#2 at 20pt \f #1} #2, Adobe #3 #4\par\medskip
}

\def\entrynontext#1#2#3#4{%#1=official ps name, #2=tex font name,
                          %#3=adobe package code, #4=adobe file name
  {\font\f=ptmrb at 10pt \f #1} (non-text) #2, Adobe #3 #4\par\medskip
}
```

Create the tex file, after setting func in fontdb.rc:

```
$ ./fontdb.rc > catalog.tex
```

Typeset the font catalog, it cannot be easier:

```
$ tex catalog.tex
```

Ghostscript must load the font file for each font used in the document, and may choke on a postscript file with two hundred fonts. A remedy is to split the postscript file in a number of smaller files, e.g., one file per page. The dvips command to accomplish this is:

```
$ dvips -i -S 1 catalog.dvi
```

This results in a number of postscript files catalog.001, catalog.002, etc, which may be printed much easier.

The font catalog gives an impression of each font, but this is not sufficient for the symbol fonts. For symbol fonts we need to print a table showing each glyph and the char codes to be used in your TEX documents. The easiest way to print the table for a font is to use the interactive latex file nfssfont:

```
$ latex nfssfont
```

Conclusion

Installing fonts doesn't have to be complicated when you know how to do it. Some scripts were shown to aid the installation. Some tips were mentioned that hint at hours of despair by the author, and he hopes that you will be able to avoid that experience.

Getting free fonts, buying more fonts, and installing those, however, is only the preparation for using your fonts. Now is the time to add a book on typography to your TEX bookshelf, and to create even more beautiful books.

CaslonTwoTwentyFour-Book pc2k8r, Adobe 950-00 c2w

CaslonTwoTwentyFour-BookIt pc2ki8r, Adobe 950-00 c2wi

CaslonOpenFace pcarl8r, Adobe 950-00 ca

CooperBlack pcbc8r, Adobe 950-00 cb

CooperBlack-Italic pcbci8r, Adobe 950-00 cbi

Cheltenham-Bold pctb8r, Adobe 950-00 chb

Cheltenham-BoldItalic pctbi8r, Adobe 950-00 chbi

Usherwood-Book puwk8r, Adobe 950-00 usw

Usherwood-BookItalic puwki8r, Adobe 950-00 uswi

ROSEWOOD-FILL prf8r, Adobe 950-00 uxfi

ROSEWOOD-REGULAR prr8r, Adobe 950-00 uxrg

Veljovic-Bold pvjb8r, Adobe 950-00 vlb

Veljovic-BoldItalic pvjbi8r, Adobe 950-00 vlbi

Veljovic-Black pvjc8r, Adobe 950-00 vlbl

fonts: tutorial

NFSS: using font families in LaTeX 2ε

Siep Kroonenberg
Kluwer Academic Publishers
Dordrecht
siepo@cybercomm.nl

abstract

This paper gives a brief overview of the LaTeX 2ε NFSS font machinery and font definition files. It also gives examples of ad-hoc font changes with low-level NFSS commands.

keywords

NFSS, font families, font attributes

Plain TeX doesn't support font attributes that you can vary independently: if you specify `\bf\it` then you get italics; it you specify `\it\bf` then you get bold. LaTeX 2.09 had the same shortcoming. The *New Font Selection Scheme*, NFSS for short, changed this: it provides mechanisms for organizing fonts into families, in which the different members are identified by different font attributes which can be modified independently.

NFSS is now integrated into LaTeX 2ε.

This paper introduces you to the low-level LaTeX font machinery.

Font attributes

A font is specified by the following attributes:

- `\fontencoding{`*encoding*`}`: which characters are available at what slots. Normally, you don't explicitly concern yourself with encodings; the format file will have picked a default, which may sometimes be overruled by your class file.
- `\fontfamily{`*family*`}`: this may be e.g. be Computer Modern or Helvetica; [5] in this journal describes the rather desperate Berry system of naming used by most TeX distributions; see also [6]. You could typeset you document in Times instead of Computer Modern by adding a command `\renewcommand{\rmdefault}{ptm}` to the preamble.
- `\fontseries{`*series*`}`: these are really two attributes rolled into one: width ((medium-)condensed–medium) and weight (light–**bold**)
- `\fontshape{`*shape*`}`: options include upright, *italic*, *slanted*, SMALL CAPS
- `\fontsize{`*point size*`}{`*baselineskip*`}`: usually specified through high-level commands `\normalsize` or `\footnotesize`

Font families

For normal text font attributes may be `{OT1}` (old encoding), `{ptm}` (Times), `{m}` (medium weight) and `{n}` (normal shape). A command `\bfseries` would change the weight attribute from medium to bold and leave the other attributes unchanged. LaTeX needs mappings from sets of attributes to actual font files. Such mappings are defined in *font definition* (`.fd`) files. One .fd file lists all fonts available for one encoding and one family. For instance, a file `ot1ptm.fd` describes all fonts belonging to the ptm (Times Roman) font family with the OT1 (Old TeX) encoding. It looks something like this:

```
\ProvidesFile{ot1ptm.fd}
   [1997/09/30 Fontinst v1.6 font
     definitions for OT1/ptm.]

\DeclareFontFamily{OT1}{ptm}{}

\DeclareFontShape{OT1}{ptm}{b}{n}{
  <-> ptmb7t}{}
\DeclareFontShape{OT1}{ptm}{b}{sc}{
  <-> ptmbc7t}{}
\DeclareFontShape{OT1}{ptm}{b}{sl}{
  <-> ptmbo7t}{}
\DeclareFontShape{OT1}{ptm}{b}{it}{
  <-> ptmbi7t}{}
\DeclareFontShape{OT1}{ptm}{m}{n}{
  <-> ptmr7t}{}
\DeclareFontShape{OT1}{ptm}{m}{sc}{
  <-> ptmrc7t}{}
\DeclareFontShape{OT1}{ptm}{m}{sl}{
  <-> ptmro7t}{}
\DeclareFontShape{OT1}{ptm}{m}{it}{
  <-> ptmri7t}{}

\DeclareFontShape{OT1}{ptm}{bx}{n}{
  <-> ssub * ptm/b/n}{}
\DeclareFontShape{OT1}{ptm}{bx}{sc}{
  <-> ssub * ptm/b/sc}{}
\DeclareFontShape{OT1}{ptm}{bx}{sl}{
  <-> ssub * ptm/b/sl}{}
```

```
\DeclareFontShape{OT1}{ptm}{bx}{it}{
  <-> ssub * ptm/b/it}{}
\DeclareFontShape{OT1}{ptm}{l}{n}{
  <-> ssub * ptm/m/n}{}
\DeclareFontShape{OT1}{ptm}{l}{sc}{
  <-> ssub * ptm/m/sc}{}
\DeclareFontShape{OT1}{ptm}{l}{sl}{
  <-> ssub * ptm/m/sl}{}
\DeclareFontShape{OT1}{ptm}{l}{it}{
  <-> ssub * ptm/m/it}{}

\DeclareFontShape{OT1}{ptm}{m}{ui}{
  <-> ssub * ptm/m/it}{}
\DeclareFontShape{OT1}{ptm}{b}{ui}{
  <-> ssub * ptm/b/it}{}

\endinput
```

The line

```
\DeclareFontFamily{OT1}{ptm}{}
```

specifies the font family to be defined by the file. This should match the filename, except for the case of the encoding.

The following line

```
\DeclareFontShape{OT1}{ptm}{b}{it}{
  <-> ptmbi7t}{}
```

associates a combination of encoding, family, weight/width and shape with a font on disk, i.e. with a `.tfm` file[1]. Because this is a scalable font, no mention needs to be made of specific sizes.

For Computer Modern, this would look a good deal messier; a fontshape definition from *e.g.* `ot1cmtt.fd` looks like

```
\DeclareFontShape{OT1}{cmtt}{m}{n}
     {%
      <5><6><7><8>cmtt8<9>cmtt9%
      <10><10.95>cmtt10%
      <12><14.4><17.28><20.74><24.88>cmtt12%
      }{}
```

The fifth parameter assigns fonts to (sets or ranges of) font sizes.

Both `\DeclareFontFamily` and `\DeclareFontShape` have a final parameter in which 'loading settings' can be given but which is usually left empty.

You may have noticed a third type of declaration above:

```
\DeclareFontShape{OT1}{ptm}{l}{n}{
  <->ssub * ptm/m/n}{}
```

says that weight 'light' (`l`) is unavailable and that weight 'medium' (`m`) should be used instead. A similar declaration substitutes bold (`b`) for bold extended (`bx`).

Pound for pound

A weird one is

```
\DeclareFontShape{OT1}{ptm}{m}{ui}{
  <-> ssub * ptm/m/it}{}
```

which says that fontshape italic (`it`) must be substituted for 'upright italic'(`ui`). Upright italic is similar to italic, but without the slant:

agi

Why bother with upright italic when nobody uses it anyway? Well, one might use it unintentionally: in OT1 encoding, dollars and pounds occupy the same slot, dollars coming from a 'Roman' font (upright or slanted), and pounds from an italic font, again, upright or slanted. This declaration should ensure that with Times in OT1 encoding upright pound symbols are rendered as italic pound symbols rather than upright dollars.

Mix and match

You may think that TeX ought to be able to figure out by itself which fonts combine into which families by looking at font names – what good would the Berry naming scheme (see e.g. [5]) be otherwise.

Well, maybe it could, but it doesn't. For the user, this means additional complexity, but also an opportunity to mix and match his own families. Here is the MAPS font family definition:

```
\DeclareFontFamily{OT1}{ptfs}{}

\DeclareFontShape{OT1}{ptfs}{b}{n}{
  <-> s * [0.95] pfrb7t}{}
\DeclareFontShape{OT1}{ptfs}{b}{sc}{
  <-> pfrbc7t}{}
\DeclareFontShape{OT1}{ptfs}{b}{it}{
  <-> s * [0.95] pfrbi7t}{}
\DeclareFontShape{OT1}{ptfs}{m}{n}{
  <-> ptmr7d}{}
\DeclareFontShape{OT1}{ptfs}{m}{sc}{
  <-> ptmrc9t}{}
\DeclareFontShape{OT1}{ptfs}{m}{it}{
  <-> ptmri7d}{}

\DeclareFontShape{OT1}{ptfs}{m}{sl}{
```

1. Actually, this is a virtual font, although that is of no concern to TeX itself. When the dvi-driver has to render the font, it will first look for a corresponding virtual font, *i.e.* a `.vf` file. This file will contain instructions to typeset the original characters, using characters from possibly different fonts at possibly different positions in the encoding vector.

```
  <-> ssub * ptfs/m/it}{}
\DeclareFontShape{OT1}{ptfs}{bx}{n}{
  <-> ssub * ptfs/b/n}{}
\DeclareFontShape{OT1}{ptfs}{bx}{it}{
  <-> ssub * ptfs/b/it}{}
\DeclareFontShape{OT1}{ptfs}{bx}{sl}{
  <-> ssub * ptfs/bx/it}{}
\DeclareFontShape{OT1}{ptfs}{b}{sl}{
  <-> ssub * ptfs/b/it}{}
\DeclareFontShape{OT1}{ptfs}{bx}{ui}{
  <-> ssub * ptfs/bx/it}{}

\DeclareFontShape{OT1}{ptfs}{b}{ui}{
  <->sub * ptfs/b/it}{}
\DeclareFontShape{OT1}{ptfs}{m}{ui}{
  <->sub * ptfs/m/it}{}
```

This family combines medium weights from Times with bold weights from Adobe Frutiger. Moreover, it scales the Frutiger fonts to 95% of their original size, to compensate for their larger x-height.

X-height refers to the height of lower-case letters without ascenders or descenders, such as 'x'. Font sizes are measured including ascenders and descenders. A font with a large x-height looks larger than a font with a small x-height at the same point size. Without such a correction, Times and Frutiger would not harmonize very well.

Ad-hoc font changes

LaTeX has been designed for structured documents. Most of the time the class file will take care of font changes, and most of the time that is a good thing.

If you use LaTeX for everything else, then you may also want to use it for one-off projects such as invitations or flyers, even though LaTeX is not the tool of choice for such documents. Here one often wants to create a headline at a size or in a font that the LaTeX 2_ε class files don't provide for.

One could use plain a TeX font commands such as

```
\font\headlinefont=bchb7t at 22pt
 :
\headlinefont Headline
```

which produces

Headline

There are problems with this approach: because it bypasses the NFSS engine, implicit font changes are not handled properly, as the following example demonstrates:

```
{\headlinefont
Head \textit{italic} $\alpha$ tail}
```

produces

Head *italic* α **tail**

The new font is picked relative to the original font. Presumably, we wanted something like

Head *italic* α tail

We could have achieved this by defining `\headlinefont` as follows:

```
\newcommand{\headlinefont}{%
 \fontfamily{bch}\bfseries
 \fontsize{22pt}{22pt}\selectfont}
```

The current MAPS classfile still uses the old OT1 encoding. Using plain TeX font commands in combination with T1 encoding can produce even nastier effects:

hei“g

instead of

heiïg

This could happen because some characters, such as the dotless *i*, come from another font.

If you bemoan the increased bulk and complexity of LaTeX 2_ε, keep in mind that the innovations in it solved some real problems.

References

1. Michel Goossens, Frank Mittelbach and Alexander Samarin, *The LaTeX Companion*, Addison-Wesley, 1994.
2. Michel Goossens, Sebastian Rahtz and Frank Mittelbach, *The LaTeX Graphics Companion*, Addison-Wesley, 1997.
3. The LaTeX3 Project Team, *LaTeX 2_ε font selection*, CTAN/`macros/latex/base/fntguide.tex`. This is the definitive reference on LaTeX 2_ε font selection.
4. Leslie Lamport, *LaTeX: A Document Preparation System*, Second Edition, Addison-Wesley, 1994.
5. Maarten Gelderman, *How to install a Type 1 font using* FONTINST, MAPS 22 (this issue)
6. Tomas Rokicky, *DVIPS manual*, distributed with DVIPS.

fonts: context

Fonts in ConTEXt

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

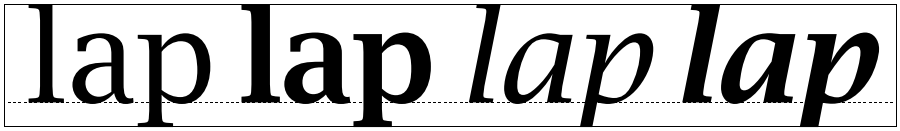
abstract
This article is the revised chapter 5 of the ConTEXt reference manual, typeset in the Maps layout. We will pay attention to all kind of font switching, fine tuning, and also provide some background information. Special attention is paid to the mapping of font names to files and some words are spent on encodings. This text is typeset in Lucida Bright but within the layout specification of the Maps.

keywords
ConTEXt, fonts, encodings, typography

1 Inleiding

We denken in veel gevallen in woorden, verbaal. Gedurende duizenden jaren heeft men getracht deze gedachten vast te leggen op een medium, bijvoorbeeld steen of papier. Klanken en grammaticale constructies worden vastgelegd in series karakters, die op hun beurt, al dan niet gecombineerd, worden weergegeven in plaatjes: glyphs. Deze afbeeldingen hebben zich niet alleen steeds verder verfijnd, ze zijn ook buitengewoon herkenbaar, zelfs als verschillende vormgevers er hun interpretatie aan geven.

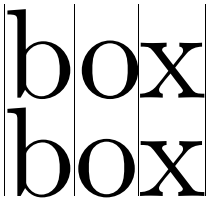
Van links naar rechts zien we hier een Computer Modern, een Lucida Bright, een Times Roman en een Antiqua Torunka font, allen geschaald naar 40pt.



Binnen een bepaald letterontwerp komen in de regel varianten voor die weliswaar dezelfde vorm hebben, maar toch wezenlijk verschillen. Hierboven zien we bijvoorbeeld een standaardvorm, een vette variant, een bijpassende italic en een vette italic, allen Lucida Bright.

We zullen in deze handleiding niet diepgaand ingaan op de wijze waarop letters worden gecombineerd tot woorden, die op hun beurt regels vormen, al dan niet met afbrekingen, samen goed voor een paragraaf. In dit hoofdstuk staat het wisselen van font centraal.

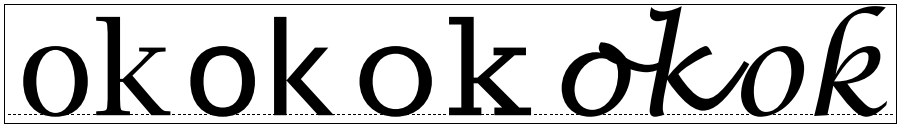
De afstand tussen de verschillende glyphs kan afhangen van de combinatie. Het volgende voorbeeld illustreert dit. Zowel de afstand tussen de b en de o als die tussen de o en de x is aangepast. We noemen dit kerning.



In dit voorbeeld tonen we een Computer Modern, het standaard TEX lettertype. Dit door Knuth ontworpen lettertype is een variant op een Monotype Times, en kent, in tegenstelling tot de Lucida die in deze handleiding wordt gebruikt, relatief veel kerning paren.

Dit soort micro–typografie valt buiten de invloedssfeer van de gebruiker, het maakt deel uit van het letter ontwerp. De gebruiker kan echter wel invloed uitoefenen op de keuze van een lettertype, de afstand tussen regels, het oogje dat TEX dichtknijpt bij het zetten van een paragraaf, en nog enkele aspecten. In dit hoofdstuk beperken we ons tot de keuze van het lettertype.

Er zijn verschillende manieren om het karakter van een lettertype te benoemen. Men kan bijvoorbeeld klassificeren naar historisch gezien de tijd waarin een letter is ontworpen, de kenmerken van het lettertype, of de gebruikswaarde, bijvoorbeeld een boek of krant.



Hier zien we naast elkaar vier gedaanten van de Lucida: de Bright, Sans, Typewriter, Handwriting en Calligraphy. Er zijn slechts weinig ontwerpen die zo veel stijlen herbergen. De Computer Modern is ook zo'n ontwerp. Bovendien onderkent dit ontwerp als een van de weinige ontwerpen nog ontwerpgrootten, iets dat vroeger heel normaal was, maar tegenwoordig een zeldzaamheid. Hieronder zien we, uitvergroot tot 48 punten, het verschil tussen een 5, 7, 9, 12 en 17 punts ontwerp.

We beperken ons hier tot de eerste drie verschijningsvormen. Als we voor een meer algemene aanduiding kiezen, komen we terecht bij de typeringen in tabel 1.

Serif	Sans	Mono
Regular	Support	Mono
Roman	Sans	Type

Tabel 1 Een globale indeling in klassen, gedemonstreerd aan een Computer Modern.

Hoewel de eerste twee reeksen beter aansluiten bij de typografische wereld, gebruiken we binnen ConTEXt vaak de laatste, dit omdat hij beter aansluit bij

de traditionele benamingen binnen plain TEX. Zo wordt \rm gebruikt om over te gaan op een roman/serif/regular stijl, en \tt voor mono gespatieerde of typewriter stijl.

In de volgende paragrafen gaan we in op de wijze waarop men stijl en/of varianten instelt. Een waarschuwing is daarbij op zijn plaats. Gezien de relatie tussen fonts, iets dat met name tot uitdrukking komt in de wiskundige modus (math mode), is het achterliggende mechanisme redelijk complex. Deze situatie wordt nog gecompliceerd door het feit dat zowel de rangschikking van de karakters die men intypt, als de ordening van de karakters in een font file (beiden aangeduid met encoding vector) roet in het eten kunnen gooien. Enige kennis van fonts en hun eigenaardigheden is in dat geval nodig.

2 Het mechanisme

Het wisselen van font wordt aangeduid als een font–switch. Omdat zo'n mechanisme onmisbaar is, is het tegelijk een van de oudste onderdelen van ConTEXt. Wel zijn in de loop der tijd met name in de wijze van definiëren uitbreidingen gerealiseerd. Bij het opzetten van het mechanisme zijn de volgende uitgangspunten gehanteerd.

- Er kan eenvoudig worden gewisseld van *stijl*, dat wil zeggen: roman (serif, regular), sans serif (support), teletype (ook wel monospaced) enz. (\rm, \ss, \tt enz.)
- Er zijn meerdere *varianten* van letters beschikbaar, zoals schuin en vet (\sl en \bf).
- Er worden verschillende *families* ondersteund, waaronder Computer Modern Roman en Lucida Bright.
- Er kan eenvoudig worden gewisseld van *korps*, waarbij standaard de korpsen tussen 8pt en 12pt beschikbaar zijn.
- Binnen een korps zijn verschillende *formaten* mogelijk, bijvoorbeeld voor sub- en superscripts. De *formaten* kunnen door elkaar worden gebruikt, met behoud van stijl, variant en familie.
- Er wordt rekening gehouden met de specifieke kenmerken van een *font*, waaronder de wijze waarop het font is gedefinieerd, zoals de encoding vector.

Letters, of beter gezegd: de grafische interpretatie van letters, zijn in verschillende korpsen (afmetingen) beschikbaar. Als eenheid wordt de punt gebruikt, de pt. De beschikbaarheid wordt geregeld in definitie–files. In beter tijden werden voor elke afmeting aparte letters ontworpen, een traditie die eigenlijk alleen is voortgezet in de standaard TEX fonts. Bij de meeste fonts zal men het moeten doen met een geschaald 10 punts ontwerp.

Om herhaald definiëren te voorkomen, zijn standaard de meest gebruikelijke afmetingen gedefinieerd: 8–14.4 punten. Wanneer men onverhoopt een nog niet gedefinieerde afmeting instelt, bijvoorbeeld 32 punt op een titelpagina, dan zal ConTEXt dit formaat zelf definiëren, binnen de randvoorwaarden van de standaard korpsomgeving. Standaard wordt met een precisie van 1 digit gewerkt (instelbaar op 0, 1 en 2). Dit voorkomt onnodig laden van formaten met slechts kleine verschillen. Uitgangspunt bij de afronding is dat iets kleiner vaak minder erg is dan wat groter.

Het in een tekst te gebruiken korps, de stijl en de familie worden ingesteld met het commando:

```
\stelkorpsin[..,...,..]

...      naam serif regular romaan sans support schreefloos mono type teletype
         handschrift calligrafie 5pt ... 12pt
```

terwijl lopende de tekst kan worden omgeschakeld met het commando:

```
\switchnaarkorps[..,...,..]

...      5pt ... 12pt klein groot globaal
```

Dit laatste commando laat de instellingen van het in hoofd en voet gebruikte korps intact. Met `klein` en `groot` gaat men over op een kleiner korps, zo klein dus, of een groter korps.

Het commando `\stelkorpsin` wordt als het goed is maar eenmaal gebruikt, in de stijldefinitie! Tussentijds wisselen vindt plaats met `\switchnaarkorps`. Haal dit niet door elkaar, anders kunnen vreemde, maar volledig legitieme, nevenverschijnselen optreden. Het eerstgenoemde commando heeft namelijk ook betrekking op de hoofd- en voetregels.

TEX zoekt de informatie over een bepaald lettertype in een file met de extensie `tfm`. Hoewel het mogelijk is dergelijke files van te voren te laden, stelt ConTEXt dit laden zo lang mogelijk uit. De reden hiervoor is dat dergelijke files per systeem kunnen verschillen.

De voor hoofd- en voetregels en voetnoten gebruikte letterformaten worden na het geven van dit commando automatisch aangepast, evenals de interlinie en de sprongen. De volgende, korte commando's, beperken zich echter tot de tekst: `\vi`, `\vii`, `\viii`, `\ix`, `\x`, `\xi` en `\xii`.

De commando's:

```
{\xii  met deze zetomgeving \par}
{\xi   is het mogelijk om   \par}
{\x    een hele redelijke   \par}
{\ix   ogentest te maken:   \par}
{\viii a x c e u i w m q p  \par}
```

De interlinie wordt automatisch aangepast, dit wordt links getoond. Rechts zien we wat er gebeurt als de interlinie niet zou zijn aangepast.

met deze zetomgeving
is het mogelijk om
een hele redelijke
ogentest te maken:
a x c e u i w m q p

met deze zetomgeving
is het mogelijk om
een hele redelijke
ogentest te maken:
a x c e u i w m q p

3 Omschakelen

Het mechanisme voor het wisselen van korps en lettertype is vrij geavanceerd en daarom niet eenvoudig uit te leggen. Er wordt onderscheid gemaakt in verschillende families. Binnen deze families onderscheiden we een of meer stijlen.

Voorbeelden van stijlen zijn: roman, sans serif en teletype. In enkele gevallen is een handschrift en/of calligrafische letter beschikbaar. Voorbeelden van varianten binnen een stijl zijn: **boldface** en *slanted*.

Er zijn verschillende manieren om van lettertype te wisselen. Zo gaat men met \ss over op een sans serif letter. Vanaf dat moment zijn commando's als \bf aangepast op deze nieuwe stijl. Als men consequent gebruik maakt van commando's als \bf en \sl, krijgt men als men bovenaan de tekst van stijl wisselt, automatisch de juiste overgangen op vet en schuin. Een (aanzienlijk) snellere manier van wisselen is ook beschikbaar in de vorm van: \ssbf, \sssl enz. Deze snelle manier maakt echter het als geheel in een andere stijl zetten van een tekst onmogelijk.

Hieronder zijn de varianten weergegeven. De afkorting sl staat voor *slanted*, it staat voor *italic* en bf voor **boldface**. Aanvullend zijn eventueel ook bs en bi beschikbaar, ofwel: ***bold slanted*** en ***bold italic***. Als een variant niet beschikbaar is, dan wordt hij op een redelijk alternatief geprojecteerd.

Met os duiden we aan dat we de medæval of old–style cijfers 139 willen in plaats van 139. De aanduiding sc staat voor SMALL CAPS. Met een x duiden we een kleiner lettertype aan, met a, b, c en d een groter. De actuele stijl wordt aangegeven met tf ofwel typeface.

```
\tfa \tfb \tfc \tfd
\tfx \bfx \slx \itx
\bf \sl \it \bs \bi \sc \os
```

Afhankelijk van de volledigheid van de definitie–files zijn ook varianten als \bfa, \bfb enz. beschikbaar.

Voor de liefhebber is er \tx beschikbaar. Dit is equivalent met \tfx, \bfx, \slx enz., afhankelijk van de actuele variant. Nog kleiner kan ook: \txx. Binnen \tx gaat nogmaals \tx over op deze nog kleinere variant.

Het frequent wisselen van letter leidt tot lange verwerkingstijden. Als geen gebruik wordt gemaakt van super- en subscripts en als men echt zeker weet welke letter men wil gebruiken, dan kan men meestal ook een variant op naam oproepen: \rmsl, \ssbf, \tttf enz.

Het wisselen van stijl vindt plaats met een van de commando's:

```
\rm \ss \tt \hw \cg
```

Als \rm gekozen is, leest CONTEXT het commando \tfd als \rmd. Alle standaardinstellingen maken gebruik van tf–instellingen en passen zich dus automatisch aan.

De verschillende commando's passen zich steeds zo goed mogelijk aan bij de op dat moment actuele instellingen van het lettertype en formaat. Bijvoorbeeld:

```
\rm test {\sl test} {\bf test} \tfc test {\tx test} {\bf test}
\ss test {\sl test \tx test} {\bf test \tx test}
```

levert:

test *test* **test** test test **test**
test test test **test** test

Als een letter niet voorhanden is, dan wordt een automatisch een acceptabel alternatief gekozen.

Op het typografische enigzins verwerpelijke onderstrepen en doorhalen gaan we hier niet in. Deze worden elders in de handleiding beschreven.

4 Letters

Een aantal commando's heeft een parameter `letter` waarmee een lettertype kan worden ingesteld. In dat geval kunnen commando's als `\sl` of `\rma` worden meegegeven, maar ook een van de volgende aanduidingen:

`normaal vet schuin vetschuin italic vetitalic type klein kleinvet kleinschuin ... kleinitalic ... kleintype kapitaal`

Hoewel flexibiliteit zijn grenzen kent, worden bij de parameter `letter` zowel `vet` als `\bf` of `bf` geaccepteerd. Zelfs de achter de schermen opererende naam `12ptrmbf` is toegestaan. Deze laatste variant wordt afgeraden, vereist inzicht in onderliggende macros, maar is wel lekker snel.

5 Beschikbare alternatieven

Er zijn slechts weinig font sets (korpsen) beschikbaar die ook wiskunde aan kunnen. Natuurlijk is er de Computer Modern Roman, maar de zeer fraaie Lucida Bright en de door uitgevers zeer gewaardeerde Times zijn ook bruikbaar. Al deze fonts beschikken over een volledige set karakters voor wiskundig zetwerk. De Computer Modern Roman onderscheidt zich zowel door volledigheid als door de hoogwaardige kwaliteit van andere fonts. Er zijn op dit ontwerp enkele aanvullingen beschikbaar: Euler en Concrete.[1]

De Computer Modern Roman bestaat uit zo'n 70 lettertypes en letterformaten. Omdat een aantal formaten niet standaard aanwezig kunnen worden geacht, zijn onder de optie `cmr` bijvoorbeeld de 11 punts letters gedefinieerd als geschaalde 9 en 10 punts letters. Met `eul` en `con` krijgen we een variant op de Computer Modern.

```
\toonkorps[..,...,..]
...      zie p 46: \stelkorpsin
```

Met behulp van het commando `\toonkorps` kan een overzicht worden gegenereerd van de beschikbare lettertypen. Hieronder is het 9pt–korps Computer Modern Roman (`cmr`) weergegeven. De oplettende lezer zal zien dat niet alle varianten (standaard) beschikbaar zijn.

[cmr,9pt]													
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfd
\rm	Ag	AG	*Ag*	*Ag*	**Ag**	***Ag***	***Ag***	Ag	Ag	Ag	Ag	Ag	Ag
\ss	Ag	Ag	*Ag*	*Ag*	**Ag**	**Ag**	**Ag**	Ag	Ag	Ag	Ag	Ag	Ag
\tt	Ag	Ag	*Ag*	*Ag*	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

1 Zie Concrete Mathematics van Knuth cs., in veel opzichten zowel typografisch als didactisch een perfect boek.

Zoals hieronder te zien is, is een 9pt Lucida Bright (`lbr`) wat groter dan een 9pt Computer Modern Roman. Een x–letter, bijvoorbeeld `\bfx` is standaard 2pt kleiner dan het op dat moment gebruikte formaat. De grotere formaten zijn met behulp van TEX's `\magstep` geschaald.

[lbr,9pt]													
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfd
\rm	Ag	AG	*Ag*	*Ag*	**Ag**	***Ag***	***Ag***	Ag	Ag	Ag	Ag	Ag	Ag
\ss	Ag	Ag	*Ag*	*Ag*	**Ag**	***Ag***	***Ag***	Ag	Ag	Ag	Ag	Ag	Ag
\tt	Ag	Ag	*Ag*	*Ag*	**Ag**	***Ag***	***Ag***	Ag	Ag	Ag	Ag	Ag	Ag

Meestal werkt men met eenzelfde familie en korps. Als men echter verschillende families door elkaar gebruikt, kunnen problemen ontstaan. Deze zijn het gevolg van de verschillende manieren waarop leveranciers de karakters coderen. Er kunnen met name problemen ontstaan rond karakters met accenten en bijzondere karakters, bijvoorbeeld wiskundige karakters.

Een opmerking tot slot. Als men een van de grotere letterformaten heeft gekozen, bijvoorbeeld `\tfb`, dan is op dat moment `\tf` gelijk aan `\tfb`, `\bf` gelijk aan `\bfb` enz. Deze werkwijze bleek in de praktijk de voorkeur te verdienen boven terugkeer naar het standaard letterformaat.

6 Benadrukken

Binnen de meeste macropaketten is wel het commando `\em` beschikbaar. Dit commando gedraagt zich als een kameleon, dat wil zeggen dat het zich aanpast aan de actuele stijl. Ook binnen ConTEXt is `\em` beschikbaar. Het commando heeft de volgende kenmerken:

- er kan worden overgegaan op *slanted* of *italic*
- er wordt binnen `\bf` overgegaan op ***bold slanted*** of ***bold italic*** (als beschikbaar)
- er vindt automatisch een zogenaamde ‚italic correction' plaats (`\/`)

De vette schuine of italic letter wordt natuurlijk alleen ondersteund als `\bs` en `\bi` als zodanig beschikbaar zijn.

```
De afkorting {\em em} staat voor {\em emphasis}.
{\em De afkorting {\em em} staat voor {\em emphasis}.}
{\bf De afkorting {\em em} staat voor {\em emphasis}.}
{\em \bf De afkorting {\em em} staat voor {\em emphasis}.}
{\it De afkorting em {\em staat \bf voor} emphasis.}
{\sl De afkorting em {\em staat \bf voor} emphasis.}
```

Het bovenstaande levert: De afkorting *em* staat voor *emphasis*. *De afkorting* em *staat voor* emphasis. **De afkorting *em* staat voor *emphasis*. *De afkorting* em *staat voor* emphasis.** *De afkorting em* staat **voor** *emphasis. De afkorting em* staat **voor** *emphasis.*

Het voordeel van het gebruik van `\em` boven `\it` en/of `\sl` ligt in het feit dat op deze manier consistentie wordt afgedwongen.

Standaard is de emphasis ingesteld op *schuin*, maar in deze tekst staat ze op *italic*. Dit is in te stellen met:

```
\stelkorpsomgevingin[default][em=italic]
```

7 Kapitalen

Woorden en afkortingen kunnen in kapitalen worden weergegeven. Zowel kleine letters als hoofdletters worden automatisch geconverteerd. Een met \kap opgeroepen kapitaal is een x-letter. Bij een overgang naar schuin (\sl), vet (\bf) enz. verandert de kapitaal mee.

```
\kap{...}
...     tekst
```

```
\Kap{...}
...     tekst
```

```
\KAP{...}
...     tekst
```

```
\Kaps{.. ... ..}
...     tekst
```

Het eerste commando converteert alle letters naar hoofdletters. Het verdient aanbeveling zelf geen hoofdletters te gebruiken, dit omdat bij echte kleinkapitalen (small caps) verschil bestaat tussen hoofd- en kleine letters.

```
Het is, gezien het grote aantal wetten dat wordt uitgevaardigd,
zeer waarschijnlijk dat de kapitaal \kap{w} het meest wordt
gebruikt. Er zijn voorbeelden te over: \kap{wvw}, \kap{ww} en
\kap{bw}. Niet iedere \kap{w} staat overigens voor een wet,
neem nu \kap{www}.
```

heeft als resultaat:

Het is, gezien het grote aantal wetten dat wordt uitgevaardigd, zeer waarschijnlijk dat de kapitaal W het meest wordt gebruikt. Er zijn voorbeelden te over: WVW, WW en BW. Niet iedere W staat overigens voor een wet, neem nu WWW.

Een \kap binnen een \kap wordt afgevangen en hoeft dus niet tot problemen te leiden:

```
\kap{Kapitaalkrachtige mensen maken niet zelden de
\kap{kapitale} fout hun \kap{kapitaal} exponentieel
te doen willen stijgen. Speculatie kan namelijk tot
\nokap{kapitaalvernietiging} leiden.}
```

of:

KAPITAALKRACHTIGE MENSEN MAKEN NIET ZELDEN DE KAPITALE FOUT HUN KAPITAAL EXPONENTIEEL TE DOEN WILLEN STIJGEN. SPECULATIE KAN NAMELIJK TOT kapitaalvernietiging LEIDEN.

We zien in dit voorbeeld dat binnen een \kap tijdelijk het in kapitalen zetten kan worden onderbroken met \nokap.

```
\nokap{...}
```

... *tekst*

Het commando \Kap maakt van de eerste letter een hoofdletter en \KAP maakt hoofdletters van de letters die worden voorafgegaan door \\. Met \Kaps ten slotte, kan men van meerdere woorden de eerste letter een hoofdletter maken.

```
\stelkapitalenin[..,..=..,..]
```

```
titel    ja nee
sc       ja nee
```

Met dit commando kunnen wat kenmerken van kapitalen worden ingesteld. De instelling sc=ja schakelt over op echte SMALL CAPS. Met titel bepalen we of in titels kapitalen worden gehonoreerd.

Naast deze \kap–commando's zijn er nog:

```
\Woord{...}
```

... *tekst*

en

```
\Woorden{.. ... ..}
```

... *tekst*

Deze commando's maken van de beginletter(s) van woorden hoofdletters. Een woord kan in zijn geheel worden omgezet in hoofdletters met:

```
\WOORD{...}
```

... *tekst*

We sluiten af met een voorbeeld van echte klein–kapitalen. Als deze beschikbaar zijn, dan is het wellicht fraaier om bij afkortingen en logo's de echte klein–kapitaal \sc te gebruiken, dan een pseudo–kapitaal \kap. Dit kan men instellen met de parameter letter.

```
Soms verwordt een afkorting tot woord zonder bepaalde
betekenis, dit geldt bijvoorbeeld voor de namen van de
zendgemachtigden {\sc vara} en {\sc vpro}. Wat heeft
{\sc Pragma} eigenlijk ooit betekend?
```

Levert:

Soms verwordt een afkorting tot woord zonder bepaalde betekenis, dit geldt bijvoorbeeld voor de namen van de zendgemachtigden VARA en VPRO. Wat heeft PRAGMA eigenlijk ooit betekend?

HET IS NATUURLIJK ALTIJD MOGELIJK EEN STUKJE TEKST IN KLEIN–KAPITALEN TE ZETTEN. BESEF ECHTER WEL DAT ONDERKAST LETTERS MEER DISCRIMINEREN EN DUS GEMAKKELIJKER LEZEN.

Een belangrijk verschil tussen \kap en \sc is dat het laatste commando staat voor een speciaal ontworpen lettertype. Het commando \kap past zich daarentegen aan het actuele lettertype aan: *KAP*, **KAP**, ***KAP*** enz.

Sommige zetpakketten rekken de woorden uit om een acceptabele uitlijning te krijgen. Dit wordt in ConTeXt bewust niet ondersteund! Wel kunnen, bijvoorbeeld in titels, woorden worden uitgerekt met:

```
\uitgerekt{...}
...     tekst
```

Er wordt bij het uitrekken uitgegaan van de actuele breedte.

```
\hbox to \hsize {\uitgerekt{hier\\zit\\veel\\rek\\in ...}}
\hbox to 20em   {\uitgerekt{... en\\hier\\wat\\minder}}
```

Met \\ geven we een spatie aan ({} mag ook).

h i e r z i t v e e l r e k i n . . .
. . . e n h i e r w a t m i n d e r

Dergelijke escapades zijn typografisch eigenlijk alleen toegestaan in koppen. De aan het bovenstaande commando ten grondslag liggende macro's lossen dit op door de tekst karakter voor karakter in te lezen en af te handelen.

8 Getypte tekst

Tekst kan in getypte vorm worden weergegeven. Daarbij wordt de indeling in regels gehandhaafd. De tekst wordt opgenomen tussen de commando's:

```
\starttypen ... \stoptypen
```

Zoals in:

```
\starttypen
In deze tekst zijn voorbeelden genoeg te vinden van getypte
tekst. De commando||definities en voorbeelden worden met
behulp van de genoemde commando's weergegeven, zo ook dit
voorbeeld.
\stoptypen
```

Voor in de tekst opgenomen typewerk is het commando \type beschikbaar.

```
\type{...}
...     tekst
```

Een file kan in getypte vorm in de tekst worden opgenomen met behulp van het commando:

```
\typefile{...}
...     file
```

Instellingen vinden plaats met:

```
\steltypenin[...][..,..=..,..]

...                 file typen naam
spatie              aan uit
pagina              ja nee
optie               schuin normaal commandos kleur geen
voor                commando
na                  commando
marge               maat standaard
evenmarge           maat
onevenmarge         maat
blanko              maat klein middel groot standaard halveregel regel
escape              /
springvolgendein    ja nee
palet               naam colorpretty
```

Deze setup beïnvloedt de display verbatim (\starttypen) en het verbatim zetten van files (\typefile) en buffers (\typebuffer). Het eerste, optionele argument kan worden gebruikt om de specifiek verbatim omgeving te specificeren.

```
\steltypenin[file][marge=standaard]
```

Als spatie=aan, dan worden de spatie's zichtbaar gemaakt:

```
Niet-uitlijnen␣verdient␣mijns␣inziens␣de␣voorkeur
boven␣␣␣uitlijnen␣␣␣met␣␣␣behulp␣␣␣van␣␣␣spaties
en␣␣z␣e␣k␣e␣r␣␣boven␣␣het␣␣oprekken␣␣van␣␣woorden
```

Een bijzonder geval is het volgende:

```
\definieertypen
  [breedtypen]

\steltypenin
  [breedtypen]
  [evenmarge=-1.5cm,onevenmarge=0cm]
```

Dit kan worden gebruikt als:

```
\startbreedtypen
Soms kan een regel verbatim erg lang zijn, en omdat we niet afbreken,
drukken we hem op de even pagina's dus maar in de marge.
\stopbreedtypen
```

Op een linker bladzijde wordt de in verbatim gezette tekst in de kantlijn gedrukt.

```
Soms kan een regel verbatim erg lang zijn, en omdat we niet afbreken,
drukken we hem op de even pagina's dus maar in de marge.
```

We kunnen in-line verbatim instellen met:

```
\steltypein[..=..]

spatie    aan uit
optie     schuin normaal geen
```

Als bij de instellingen optie de waarde schuin heeft, wordt alle tekst tussen << en >> in een *schuine typeletter* gezet. Dit kan bij alle hierboven genoemde commando's. Zo levert \type{aa<<bb>>cc} op: aa*bb*cc.

Omwille van de leesbaarheid kan in plaats van de *buitenste* haakjes ook gebruik worden gemaakt van andere karakters dan { en }. Er kan een eigen, niet actief, karakter worden gekozen, bijvoorbeeld: \type+ziezo+ of \type-ziezo-. Ook kunnen de reeds genoemde << en >> worden gebruikt: \type<<ziezo>> of zelfs \type<ziezo>.

De instelling optie=commandos maakt het mogelijk in een getypte tekst commando's uit te voeren. Bij deze optie wordt \ vervangen door /.

Ook deze optie is met name bedoeld voor het opstellen van handleidingen, bijvoorbeeld als een toelichting moet worden opgenomen:

```
\ziezo <</rm : dit commando doet niets>>
 /vdots
\ozeiz <</sl : en dit commando ook niet>>
```

De dubbele << en >> vervullen hier de functie van {}.

Binnen de type–commando's wordt gebruik gemaakt van \tttf. Als we \tt hadden gebruikt, dan zou \sl een schuine en \bf een vette typeletter opleveren. Nu gebeurt dit dus niet:

\ziezo : dit commando doet niets

⋮

\ozeiz *: en dit commando ook niet*

Wellicht de meest interessante optie betreft het verbatim zetten van een programma source. We beperken ons hier tot het verschijnsel, en verwijzen voor meer details naar de documentatie in de files verb-*xxx*.tex en cont-ver.tex. In de laatst genoemde file vinden onder meer we de regels:

```
\definieertypen [MP]  [optie=MP]
\definieertypen [PL]  [optie=PL]
\definieertypen [JS]  [optie=JS]
\definieertypen [TEX] [optie=TEX]
```

Allereerst zien we dat het mogelijk is een eigen verbatim omgeving te definiëren. Dit gebeurt met het commando:

```
\definieertypen[...][..,..=..,..]

...      file typen naam
..=..    zie p 53: \steltypenin
```

De bovenstaande definities koppelen zo'n omgeving aan een optie.

```
\startMP
beginfig (12) ;
  MyScale = 1.23 ;
  draw unitsquare scaled MyScale shifted (10,20) ;
endfig ;
\stopMP
```

Dit ziet er gezet uit als:

```
beginfig (12) ;
  MyScale = 1.23 ;
  draw unitsquare scaled MyScale shifted (10,20) ;
endfig ;
```

Deze opties zorgen er dus voor dat de tekst wordt gezet conform de ingestelde taal. Het is mogelijk meerdere filters te schrijven, standaard worden MetaPost en MetaFont ondersteund, PERL, JAVASCRIPT, en natuurlijk TEX. De wijze van weergeven is gekoppeld aan het kleurmechanisme, waarbij gebruik wordt gemaakt van paletten; vandaar de instelling `palet` in `\steltypenin`.

Tot slot bieden nog wat alternatieven voor `\type`. De met dit commando weergegeven woorden worden niet afgebroken. Wil men dit toch, dat is er:

```
\typ{...}
...     tekst
```

Moch men onverhoopt de behoefte hebben een handleiding over TEX te willen zetten, dan kunnen de volgende twee commando's van pas komen:

```
\tex{...}
...     tekst
```

```
\arg{...}
...     tekst
```

Het eerste plaatst een \ voor de getypte tekst, het tweede omringt de tekst met {}.

9 Wiskunde

Veel van de TEX gebruikers hebben voor dit programma gekozen omdat het zo goed is in wiskundig zetwerk. De betrokkenheid van TEX op wiskundige typografie heeft zijn stempel gedrukt op het font mechanisme. We zullen de lezer niet belasten met details, maar centraal element is de familie. Elke variant vormt een eigen familie. Er is dus een familie voor `\bf`, `\it`, enz. Binnen een familie onderscheiden we drie leden: text, script en scriptscript, ofwel normaal, kleiner, nog kleiner. De normale afmetingen worden gebruikt voor de lopende tekst, de kleinere voor bijvoorbeeld superscripts. Het volgende voorbeeld laat zien wat de verschillende varianten doen.

```
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\rm 6x^2$
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\tf 6x^2$
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\bf 6x^2$
$\tf x^2+\bf x^2+\sl x^2+\it x^2+\bs x^2+ \bi x^2 =\sl 6x^2$
```

Gezet wordt dit:

$x^2\ \mathcal{C}\ \mathbf{x^2}\ \mathcal{C}\ x^2\ \mathcal{C}\ x^2\ \mathcal{C}\ \mathbf{x^2}\ \mathcal{C}\ \boldsymbol{x^2}\ \mathcal{D}\ 6x^2$
$x^2\ \mathcal{C}\ \mathbf{x^2}\ \mathcal{C}\ x^2\ \mathcal{C}\ x^2\ \mathcal{C}\ \mathbf{x^2}\ \mathcal{C}\ \boldsymbol{x^2}\ \mathcal{D}\ 6x^2$
$x^2\ \mathcal{C}\ \mathbf{x^2}\ \mathcal{C}\ x^2\ \mathcal{C}\ x^2\ \mathcal{C}\ \mathbf{x^2}\ \mathcal{C}\ \boldsymbol{x^2}\ \mathcal{D}\ \mathbf{6x^2}$
$x^2\ \mathcal{C}\ \mathbf{x^2}\ \mathcal{C}\ x^2\ \mathcal{C}\ x^2\ \mathcal{C}\ \mathbf{x^2}\ \mathcal{C}\ \boldsymbol{x^2}\ \mathcal{D}\ 6x^2$

We zien dat de karakters zich aanpassen, maar dat de symbolen in hetzelfde font worden gezet. Technisch gezien worden ze gezet in het lettertype dat is toegekent aan familie 0 (er zijn er maximaal 16), en dat is in ons geval standaard `\tf`.

Dat het ook anders kan, toont het volgende voorbeeld. We zien in dit voorbeeld een nieuw commando opduiken: `\mf`, wat staat voor *math font*. Dit commando zorgt ervoor dat de symbolen worden gezet in het laatst gekozen font.

$x^2\ C\ x^2\ C\ x^2\ C\ x^2\ C\ x^2\ C\ x^2\ \mathcal{D}\ 6x^2$
$\mathbf{x^2\ C\ x^2\ C\ x^2\ C\ x^2\ C\ x^2\ C\ x^2\ \mathcal{D}\ 6x^2}$
$x^2\ C\ x^2\ C\ x^2\ C\ x^2\ C\ x^2\ C\ x^2\ \mathcal{D}\ 6x^2$
$\mathbf{x^2\ C\ x^2\ C\ x^2\ C\ x^2\ C\ x^2\ C\ x^2\ \mathcal{D}\ 6x^2}$
$\mathit{x^2\ C\ x^2\ C\ x^2\ C\ x^2\ C\ x^2\ C\ x^2\ \mathcal{D}\ 6x^2}$
$\mathbf{\mathit{x^2\ C\ x^2\ C\ x^2\ C\ x^2\ C\ x^2\ C\ x^2\ \mathcal{D}\ 6x^2}}$

Met moet er overigens rekening mee houden dat in veel gevallen TEX de formule als geheel zet, en niet per se van voor naar achter werkt. Instellingen aan het eind kunnen dan ook doorwerken naar het begin.

```
$\tf\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\bf\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\sl\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\bs\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\it\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
$\bi\mf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = 6x^2$
```

De exacte positie van `\mf` is dus niet zo belangrijk, we hadden ook kunnen zeggen:

```
$\bf x^2 + x^2 + x^2 + x^2 + x^2 + x^2 = \mf 6x^2$
```

Een ander aspect van fonts in wiskundige mode betreft gereserveerde namen als sin en cos.

```
$\bf x^2 + \hbox{whatever} + \sin(2x)$
```

In tegenstelling tot plain TEX wordt hier de **sin** ook vet gezet.

$\mathbf{x^2\ C\ whatever\ C\ sin(2x)}$

De 12 punts wiskundige (Computer Modern) fonts zijn gedefinieerd met:

```
\definebodyfont [12pt] [mm]
  [ex=cmex10 at 12pt,
   mi=cmmi12,
   sy=cmsy10 at 12pt]
```

Het is mogelijk binnen math mode een andere `tf`, `bf` enz. te gebruiken.

```
\definebodyfont [10pt,11pt,12pt] [mm]
  [tf=Sans          sa 1,
   bf=SansBold      sa 1,
   sl=SansItalic    sa 1,
   ex=MathExtension sa 1,
   mi=MathItalic    sa 1,
   sy=MathSymbol    sa 1]

\stelkorpsin
```

Het eerdere voorbeeld wordt dan:

$\mathbf{x^2\ C\ whatever\ C\ sin(2x)}$

10 Em en Ex

Bij het opgeven van maten maken we onderscheid tussen fysieke eenheden, zoals `pt` en `cm` en de interne eenheden `em` en `ex`. De laatstgenoemden zijn gerelateerd aan het actuele lettertype. Gebruik van interne eenheden voorkomt vaak veel rekenwerk, omdat TEX als het ware zelf vaststelt hoe breed of hoog iets moet zijn. Enig inzicht in deze maten kan echter geen kwaad. Zo komt een `em` niet overeen met de breedte van een M, maar van een — (een em-dash). Als dit karakter niet beschikbaar is, geldt een andere waarde. Tabel 2 toont enkele voorbeelden. We zien dat de breedte van een cijfer `.5em` is (of omgekeerd: `1em` heeft de breedte van twee cijfers).

\tf	\bf	\sl	\tt	\ss	\tfx
12	**12**	*12*	12	12	12
M	**M**	*M*	M	M	M
| |	| |	| |	| |	| |	| |
├─┤	├─┤	├─┤	├---┤	├─┤	├─┤

Tabel 2 De breedte van een `em`.

Waar de `em` meestal in de breedte wordt gebruikt, gebruiken we `ex` in de hoogte. Tabel 3 toont enkele voorbeelden. We zien dat `1ex` redelijk overeenkomt met de hoogte van een x, in dit geval dus geen kapitaal.

\tf	\bf	\sl	\tt	\ss	\tfx
x	**x**	*x*	x	x	x

Tabel 3 De hoogte van een `ex`.

11 Definities

Deze paragraaf is alleen bedoeld voor nieuwsgierige gebruikers of gebruikers die willen experimenteren met het instellen van korpsen en lettertypes. Een aantal zaken worden hier niet besproken, zoals de precieze definities van accenten en encodings. Hiervoor verwijzen we naar de voorbeelden in de broncode en de files `font-xxx`.

We hebben reeds gezien dat binnen een gekozen korps allerlei kleinere en grotere afmetingen voorkomen. Deze relaties zijn vastgelegd met:

```
\definieerkorpsomgeving
  [12pt]
  [        text=12pt,      Wiskunde maten: normale afmetingen,
         script=9pt,       super- en subscripts en
   scriptscript=7pt,       supersuper- en subsubscripts.
              x=10pt,      Pseudo kapitalen en
             xx=8pt,       geneste pseudo kapitalen
          groot=12pt,      Voor geval we overgaan op groot
          klein=10pt]      of klein.
```

Wanneer we om een korpsgrootte vragen die niet op deze manier is voorgedefinieerd, dan gelden verhoudingen die overeenkomen met de bovenstaande.

Men kan deze overigens aanpassen door in plaats van bijvoorbeeld een korpsgrootte het trefwoord `default` mee te geven. Als men 'even snel' een groot korps wil definiëren, dan kan worden volstaan met:

```
\definieerkorpsomgeving [24pt]
```

In alle gevallen waarin de gebruiker niets heeft gedefinieerd, wordt teruggevallen op redelijke default waarden. Zo is het mogelijk om te schakelen naar een 12.4 korps(omgeving), zonder dat daar bijzondere voorinstellingen voor nodig zijn. Wanneer dit binnen een groep gebeurt, dan hebben de instellingen een tijdelijk karakter. Wordt een korps vaker gebruikt, dan is het verstandig dit te definiëren aan het begin van de file.

Een overzicht van de verschillende samenhangende formaten binnen een familie kan worden opgevraagd met:

```
\toonkorpsomgeving[..,...,..]

...      zie p 46: \stelkorpsin
```

Voor `lbr` levert dit commando het volgende overzicht op:

[lbr]						
text	**script**	**scriptscript**	**x**	**xx**	**klein**	**groot**
14.4pt	11pt	9pt	12pt	10pt	12pt	14.4pt
12pt	9pt	7pt	10pt	8pt	10pt	14.4pt
11pt	8pt	6pt	9pt	7pt	9pt	12pt
10pt	7pt	5pt	8pt	6pt	8pt	12pt
9pt	7pt	5pt	7pt	5pt	7pt	11pt
8pt	6pt	5pt	6pt	5pt	6pt	10pt
7pt	6pt	5pt	6pt	5pt	5pt	9pt
6pt	5pt	5pt	5pt	5pt	5pt	8pt
5pt	5pt	5pt	5pt	5pt	5pt	7pt
4pt	4pt	4pt	4pt	4pt	4pt	6pt
22pt	15.3pt	11pt	17pt	13.2pt	17pt	22pt
17pt	11.8pt	8.5pt	13.6pt	10.2pt	13.6pt	20.3pt

Voor alle gangbare formaten zijn omgevingen gedefinieerd die in de praktijk goed voldoen. Een beginnend gebruiker zal hier dus niets mee van doen hebben. Moch men toch wat willen aanpassen, dan is er:

```
\stelkorpsomgevingin[...][..,..=..,..]

...      zie p 46: \stelkorpsin
..=..    zie p 46: \stelkorpsin
```

Het eigenlijke definiëren, dat wil zeggen, het koppelen van commando's aan font files, kan op verschillende manieren gebeuren. Het meeste inzicht verkrijgt men door te kijken naar een file als `font-phv`.

```
\definefontsynonym [Sans]            [Helvetica]
\definefontsynonym [SansBold]        [Helvetica-Bold]
\definefontsynonym [SansItalic]      [Helvetica-Oblique]
\definefontsynonym [SansSlanted]     [Helvetica-Oblique]
\definefontsynonym [SansBoldItalic]  [Helvetica-BoldOblique]
\definefontsynonym [SansBoldSlanted] [Helvetica-BoldOblique]
\definefontsynonym [SansCaps]        [Helvetica]

\definebodyfont
  [14.4pt,12pt,11pt,10pt,9pt,8pt,7pt,6pt,5pt] [ss]
  [default]
```

Met `\definefontsynonym` koppelen we een logische aanduiding, zoals `SansBold` aan een font naam, zoals `Helvetica-Bold`. De eigenlijke koppeling aan een filenaam vindt elders plaats, standaard in file `font-fil` als:

```
\definefontsynonym [Helvetica-Bold] [hvb] [encoding=texnansi]
```

Dit is eigenlijk de enige plek waar een systeemafhankelijke instelling plaatsvindt. Als we onder het Karl Berry regime werken, dan ligt de volgende instelling meer voor de hand (zie `font-ber`).

```
\definefontsynonym [Helvetica-Bold] [phvb]
```

Het op die manier aan elkaar koppelen van fonts kent nauwelijks grenzen. Men mag zo vaak afbeelden als nodig. Het is leerzaam eens in `font-unk` te kijken, waar de verschillende stijlen zo worden afgebeeld dat ze door elkaar te gebruiken zijn.

```
\definefontsynonym [Regular] [Serif]
\definefontsynonym [Roman]   [Serif]
```

We zien dat de basisaanduiding `Serif` is. De default serif fonts zijn gedefinieerd met:

```
\definebodyfont [default] [rm]
  [ tf=Serif        sa 1,
   tfa=Serif        sa a,
      ...
    sl=SerifSlanted sa 1,
   sla=SerifSlanted sa a,
      ...]
```

Wat hier gebeurt is het volgende. We zagen reeds dat `\tf` het standaard lettertype is. Hier wordt `\tf` gedefinieerd als `Serif sa 1` wat betekent dat het een serif font is, geschaald op de natuurlijke korpsgrootte. Die `Serif` is elders geprojecteerd op bijvoorbeeld `LucidaBright` die op haar beurt wordt afgebeeld op de filenaam `lbr`.

Dit soort in–een–klap definities, met gebruik van `default`, maken het mogelijk snel en comfortabel een korps te definiëren. We gaan daarbij volledig voorbij aan het gegeven dat fonts een ontwerpgrootte hebben, en het geval wil dat TEX fonts die hebben. Aangezien wij, net als de meeste TEX gebruikers, begonnen zijn met de TEX fonts, biedt ConTEXt vanzelfsprekend de mogelijkheid tot zeer nauwkeurige definities. Daar wordt dan ook van gebruik gemaakt in de file `font-cmr`:

```
\definebodyfont [12pt] [rm]
  [ tf=cmr12,
   tfa=cmr12 scaled \magstep1,
   tfb=cmr12 scaled \magstep2,
   tfc=cmr12 scaled \magstep3,
   tfd=cmr12 scaled \magstep4,
    bf=cmbx12,
    it=cmti12,
    sl=cmsl12,
    bi=cmbxti10 at 12pt,
    bs=cmbxsl10 at 12pt,
    sc=cmcsc10 at 12pt]
```

We gebruiken hier de standaard TEX–specificaties `scaled` en `at`, maar we zagen reeds dat CONTEXT als aanvulling een combinatie van beiden biedt: `sa` (scaled at). Als we geen gebruik maken van de default definitie, luidt de definitie van de Helvetica bijvoorbeeld:

```
\definieerkorps [12pt,11pt,10pt,9pt,8pt] [ss]
  [tf=hv  sa 1.000,
   bf=hvb sa 1.000,
   it=hvo sa 1.000,
   sl=hvo sa 1.000,
  tfa=hv  sa 1.200,
  tfb=hv  sa 1.440,
  tfc=hv  sa 1.728,
  tfd=hv  sa 2.074,
   sc=hv  sa 1.000]
```

We geven in dit geval de schaalwaarde ten opzichte van de korpsgrootte op. Analoog aan TEX's `\magstep` kunnen we hier `\magfactor` gebruiken: in plaats van `sa 1.440` kan dus `sa \magfactor2` worden opgegeven. Omdat dit soort getallen niet alleen vervelend is, maar ook onnodig geheugen gebruikt, mag men `1.200` vervangen door `a`, enz. Deze relatie is in te stellen in de korpsomgeving.

```
\definieerkorps [12pt,11pt,10pt,9pt,8pt] [ss]
  [tf=hv sa 1,
   tfa=hv sa a, tfb=hv sa b, tfc=hv sa c, tfd=hv sa d]
```

Als font files in alle interfaces worden gebruikt, gebruiken we de engelse commando's. De definities vinden plaats in files met de naam `font-???.tex`, zie bijvoorbeeld de file `font-cmr.tex`.

De instellingen `ex`, `mi`, `sy`, `ms`, `mb` en `mc` hebben betrekking op wiskundige karaktersets. De eerste drie vinden we ook in Plain TEX, de laatste drie zijn nodig bij andere families. Zo zijn de binnen $\mathcal{AMS}$-TEX te gebruiken letters en symbolen ook binnen CONTEXT te gebruiken: `\definieerkorps[ams]`. Deze zijn ondergebracht in `ma` en `mb`.

```
\definieerkorps[..,.1.,..][.2.][..,..=..,..]

.1.      5pt ... 12pt default
.2.      rm ss tt mm hw cg
tf       file
bf       file
sl       file
it       file
bs       file
bi       file
sc       file
ex       file
mi       file
sy       file
ma       file
mb       file
mc       file
```

Men is overigens niet gebonden aan a–d. Bij wijze van voorbeeld defini‰ren we een grote variant van \tf:

```
\definieerkorps [9pt,10pt,11pt,12pt] [rm]
  [tfe=ComputerModern at 36pt]
\tfe Grote Woorden.
```

Deze ziet er als volgt uit:

Grote Woorden.

Nu we toch bezig zijn met definiëren, men kan ook onafhankelijk van het besproken mechanisme direct een font definiëren.

```
\definieerkorps[KopLetter][Regular sa 1.2]
```

Hierna kan men \KopLetter gebruiken om van font te wisselen. Zo nodig moet men de spatiëring (interlinie) instellen met \stelinterliniein, dus:

```
\KopLetter \stelinterliniein tekst \par
```

Voor gevorderde TEX–gebruikers is het dimensie–register \korpsgrootte beschikbaar. Dexe variabele kan worden gebruikt om breedtes in te stellen. Het aantal (afgeronde) punten is beschikbaar in \korpspunten.

Tot nu toe zijn we er van uitgegaan dat een a te voorschijn komt als een a. Dit is geen vanzelfsprekendheid in geval van bijvoorbeeld een ä of æ. Dit karakter is namelijk niet in ieder font aanwezig, zeker niet in de Computer Modern Typefaces. Vaak zal een combinatie van letters, zoals \"a, of een commando \ae gebruikt worden om zo'n karakter te zetten. In een aantal gevallen zal TEX uit zichzelf karakters combineren, zoals fl tot fl en niet fl. Een ander probleem vormt het omzetten van hoofd- in kleine letters en omgekeerd. Om met dit laatste te beginnen, hier is een voorbeeld van de texnansi mapping:

```
\startmapping[texnansi]
  \definecasemap 228 228 196  \definecasemap 196 228 196
  \definecasemap 235 235 203  \definecasemap 203 235 203
  \definecasemap 239 239 207  \definecasemap 207 239 207
  \definecasemap 246 246 214  \definecasemap 214 246 214
```

```
  \definecasemap 252 252 220  \definecasemap 220 252 220
  \definecasemap 255 255 159  \definecasemap 159 255 159
\stopmapping
```

Dit betekent zoveel als: karakter met code 228 wordt in geval van hoofdletters karakter 228 en wordt in kleine letters karakter 196.

Deze definities zijn te vinden in `enco-ans`. Verder vinden we in die file:

```
\startencoding[texnansi]
  \defineaccent " a 228
  \defineaccent " e 235
  \defineaccent " i 239
  \defineaccent " o 246
  \defineaccent " u 252
  \defineaccent " y 255
\stopencoding
```

en wat verderop:

```
\startencoding[texnansi]
  \definecharacter ae 230
  \definecharacter oe 156
  \definecharacter o  248
  \definecharacter AE 198
\stopencoding
```

Zoals gezegd, accenten vormen (niet alleen in TEX) een geval apart. Dit is mede een gevolg van de wijze waarop accenten worden geplaatst. Er zijn grofweg twee methoden: · f TEX plaatst de accenten zelf · f er worden karakters gebruikt waarop de accenten al staan. De bovenstaande definities zorgen ervoor dat dit alles goed gaat. Overigens gaan aan dergelijke definities soms andere vooraf, zie daarvoor de file `enco-ini`.

We keren nog even terug naar de fontdefinities. Het vlot wisselen van stijl kan ook met commando's als `\xii` of `\twelvepoint`, waarmee we voortbouwen op plain TEX. Dergelijke commando's worden gedefinieerd met:

```
\definieerfontsynoniem [twelvepoint] [12pt]
\definieerfontsynoniem [xii]         [12pt]
```

De trefwoorden bij `\stelkorpsin` zijn gedefinieerd in de trant van:

```
\definieeralgemenestijl [rm,romaan,serif,regular]     [rm]
\definieeralgemenestijl [ss,schreefloos,sans,support] [ss]
\definieeralgemenestijl [tt,teletype,type,mono]       [tt]
\definieeralgemenestijl [hw,handschrift]              [hw]
\definieeralgemenestijl [cg,calligrafie]              [cg]
```

Bij veel elders beschreven instellingen komen we `letter` of `kopletter` tegen. In die situaties kunnen we een aanduiding meegeven. Met `\definieerletter` zijn deze aanduidingen vastgelegd. Het derde argument betreft de betekenis in titels van hoofdstukken, paragrafen enz. Alleen `\kap` heeft daar betekenis.

```
\definieerletter [normaal]            [\tf]  []
\definieerletter [vet]                [\bf]  []
\definieerletter [type]               [\tt]  []
\definieerletter [italic]             [\it]  []
\definieerletter [schuin]             [\sl]  []
```

```
\definieerletter [vetitalic,italicvet] [\bs]  []
\definieerletter [vetschuin,schuinvet] [\bs]  []
\definieerletter [klein,kleinnormaal]  [\tfx] []
                 ...
\definieerletter [kap,kapitaal]        [\kap] [\kap]
```

We hebben in paragraaf 6 al aangegeven hoe benadrukken kan worden ingesteld. Met oldstyle getallen ligt dat wat anders. Er is op voorhand niet aan te geven waar die te vinden zijn. Standaard is de instelling:

```
\definefontsynonym [OldStyle] [MathItalic]
```

ofwel, ze worden uit hetzelfde font gehaald als de wiskundige italic karakters.

In deze paragraaf zagen we een mengelmoes aan engelse en nederlandse commando's. We raden aan in geval van een internationale stijldefinitie, de engelse commando's te gebruiken. Veel van de voorbeelden zijn ontleend aan files die deel uitmaken van de ConTEXt *distributie, en zijn derhalve achter de schermen engels.*

Er zijn naast de hier besproken commando's nog andere beschikbaar, zoals macro's om accenten te manipuleren. Deze worden in de file `font-ini` *besproken. Ook kan men nadere informatie vinden in* `core-fnt` *en allerlei geintjes in* `supp-fun`*. Genoeg voer dus voor liefhebbers!*

12 Omhullende tekst

We maken onderscheid tussen lopende en omhullende tekst. Onder de omhullende tekst verstaan we hoofd- en voetregels en interactieve elementen, zoals menu's. Deze zin maakt deel uit van de lopende tekst. Al het voorgaande had voornamelijk betrekking op de lopende tekst. De lettertypen van de omhullende tekst worden ingesteld met verschillende commando's. Meestal zal dit in termen van `letter=vet` gebeuren, maar instellingen als `letter=\ss\bf` zijn ook toegestaan. Instellingen als `letter=\ssbf` liggen minder voor de hand, omdat in dat geval `\kap` en dergelijke niet correct werken.

Het wisselen van stijl (`\ss`) kost tijd. Meestal is dit geen probleem, maar wanneer we bijvoorbeeld interactieve menu's gebruiken met tientallen items, dan heeft het wisselen merkbaar invloed. In dat geval is een meer effici‰nte wijze van wisselen mogelijk:

```
\stellayoutin[letter=\ss]
```

Aanvullende instellingen vinden vervolgens plaats met de betreffende commando's en de parameter `letter`, bijvoorbeeld:

```
\stelvoetin[letter=vet]
```

Overigens gelden voor de omhullende tekst altijd de instellingen van `\stelkorpsin`, ook als de lopende tekst daarvan afwijkt.

13 Files

In tabel 4 zijn enkele van de standaard meegeleverde font definitie files opgenomen. Deze worden dus aangeroepen op de drie laatste letters.

De `ans` en `il2` encoding files zijn voorgeladen. Men kan een extra encoding laden met `\useencoding`. De twee in tabel 5 laatst genoemde files hebben betrekking op het direct ondersteunen van andere dan de standaard toetsenbord karakters.

font-cmr	Computer Modern Roman
font-csr	Computer Slavik Roman (?)
font-con	Concrete Roman
font-eul	Euler
font-ams	American Mathematics Society
font-ant	Antykwa Torunska
font-lbr	Lucida Bright
font-pos	Base PostScript Fonts
font-ptm	Times Roman
font-phv	Helvetica
font-pcr	Courier
font-fil	Standard Filenames
font-ber	Karl Berry FileNames

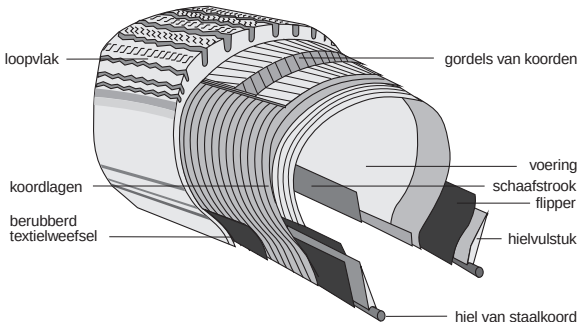
Tabel 4 Enkele standaard meegeleverde font definitie files.

enco-ans	TeXnansi
enco-il2	ISO Latin 2
enco-ibm	default IBM PC code page
enco-win	default MS Windows code page

Tabel 5 Enkele standaard meegeleverde encoding definitie files.

14 Figuren

Tot slot nog dit. Als in een tekst figuren worden opgenomen, dan ontkomt men er vaak niet aan in die figuren tekst op te nemen. Meestal zijn daarbij de TEX-fonts niet beschikbaar. Als in de tekst zelf een serif wordt gebruikt, kan bijvoorbeeld in een figuur uitstekend een Helvetica worden gebruikt. In figuur 1 is bijvoorbeeld gebruik gemaakt van een Helvetica. In het onderschrift gebruiken we Knuth's Sans Serif.



Figuur 1 Het gebruik van lettertypes in figuren.

fonts

De Euro in TEX

Hein Laan

Inleiding

Begin dit jaar is de nieuwe europese munt – de euro – officieel in gebruik genomen. Een nieuwe munt met daarbij een nieuw symbool dat zijn plaats moet krijgen tussen alle bekende letters en tekens. Het zal nog even duren eer de euro als symbool zijn vanzelfsprekende plaats heeft gevonden binnen de fonts die we in TEX gebruiken, zoals het symbool voor het Engelse pond of het dollarteken dat in veel lettertypen al lang tot de standaardverzameling hoort. Het aantrekkelijke van deze nieuwe situatie is dat er vele euro's zijn... In dit artikel bekijken we er een aantal, ook wordt verteld waar de diverse euro's zijn te vinden en hoe ze kunnen worden geïnstalleerd.

Euro's te kust en te keur

Zoals gebruikelijk bij mogelijke problemen voor TEX gebruikers, is er meer dan één oplossing te vinden. Verschillende TEXneuten hebben zich ingezet voor het beschikbaar stellen van eurosymbolen. We bekijken achtereenvolgens:

- textcompanion fonts (EC fonts)
- `eurosym` package
- `marvosym` package
- Adobe eurofont
- Linotype eurofont (commercieel)

De eerste twee opties gebruiken metafont als taal om het nieuwe symbool te beschrijven, de laatse drie zijn PostScript fonts. Het gebruik van metafont heeft als voordeel dat je het resultaat ook zonder PostScript printer makkelijk kunt gebruiken. Ook PostScript fonts zijn te gebruiken op niet-PostScript printers, maar daarvoor is weleens wat extra moeite nodig. De PostScript fonts bieden weer voordelen als je TEX gebruikt voor het maken van PDF bestanden.

textcompanion fonts (EC fonts)

De meest voor de handliggende mogelijkheid zijn de textcompanion fonts van Jörg Knappen. Om het eurosymbool uit deze fonts te gebruiken is het `textcomp` package nodig. Deze is aanwezig in een recente installatie van LATEX (Januari 1998 of later). Ik heb het onder andere getest met versie 4 van de 4TEX CD, en daarbij bleek dat een nieuwere `textcomp.sty` nodig was uit de `unpacked` directory op CTAN. Ook de `ts1enc.def` uit die directory bleek onontbeerlijk. Het kan zijn dat ook deze combinatie nog kleine problemen oplevert samen met de oudere LATEX van 4TEX versie 4 en dan is het nodig om de drie `\UndeclareTextCommand...` regels onderaan `textcomp.sty` weg te halen of elk met een % onschadelijk te maken.

De `unpacked` directory is te vinden in het CTAN archief:

```
<CTAN>:macros/latex/unpacked/...
```

Ik ga ervanuit dat de meeste mensen een nieuwe LATEX kunnen installeren, zoniet, dan moet de locale TEX goeroe te hulp geroepen worden!

Om het package te kunnen gebruiken hebben we ook nog de fonts nodig. Deze zijn ook te vinden in het CTAN archief: `<CTAN>:fonts/ec`. We vinden hier drie subdirectories, waarvan alleen `/src` strict noodzakelijk is, met behulp van de files hierin kunnen de andere twee, namelijk `ready-mf` en `tfm` gegenereerd worden. Hoe dat in zijn werk gaat staat beschreven in de file `00inst.txt` in `src`, deze aanpak scheelt tijd voor downloaden – als je de CD niet hebt – maar is niet aan te raden voor mensen die het installeren van TEX packages toch al moeilijk vinden. Het is veel eenvoudiger de directories `ready-mf` en `tfm` gewoon uit het CTAN archief te plukken.

De installatie van de fonts is – net als het installeren van LATEX – kinderlijk eenvoudig. Aangezien er een behoorlijk aantal verschillende TEX installaties bestaan ga ik ervan uit dat een installatie gebruikt wordt die conform is aan TDS (TEX Directory Structure)

De `TFM` files uit de directory `tfm` moeten op een plaats geïnstalleerd worden waar TEX ze kan vinden. Daartoe stoppen we ze in een directory:

```
texmf/fonts/tfm/jknappen/ec
```

De metafont sources (uit `src` en `ready-mf`) moeten door metafont gevonden kunnen worden. Stop de inhoud van genoemde directories in

```
texmf/fonts/source/jknappen/ec
```

Door `\usepackage{textcomp}` in de preamble van het document te plaatsen kan het eurosymbool gebruikt worden. Hiervoor wordt de macro `\texteuro` gebruikt.

En dan volgt nu een stukje (fictieve) tekst waarin het eurosymbool uit de text-companionfonts gebruikt wordt:

> Gaan we uit van een dergelijke indexatie en een verruiming van € 80.000 tot € 150.000, dan resteert er van de binnen te slepen € 850.000 nog een bedag van € 700.000.

Lelijk hè?! Maar niet getreurd! Er zijn nog meer opties...

Het Eurosym package

Een andere package die ook van metafont gebruikt is het eurosym package van Henrik Theiling.

De auteur heeft gepoogd zijn eurosymbool te laten voldoen aan de richtlijnen van de europese commisie – jawel, er zijn officiële richtlijnen uitgevaardigd door de politiek over het uiterlijk van een glyph! Het eerder besproken text companion font voldoet *niet* aan deze richtlijnen en verdient dus de voorkeur bij subsidieaanvragen (geintje!).

Om het Eurosym package te kunnen gebruiken zoeken we het weer op CTAN en we vinden het in
`<CTAN>:fonts/eurosym`

Installeren doen we door de macrofile (`eurosym.sty`) die zich bevindt in .../`eurosym/sty` weer daar te plaatsen waar TEX hem vinden kan:

`texmf/tex/latex/contrib/misc`

Het zelfde voor de font metrics uit de subdirectory .../`tfm` van `/eurosym`:

`texmf/fonts/tfm/theiling/eurosym`

En tenslotte de metafont sources uit de directory .../`src`

`texmf/fonts/source/theiling/eurosym`

Met `\usepackage[`*options*`]{eurosym}` kunnen we het symbool gaan gebruiken. Het package voorziet ons van de volgende macros:

`\euro`	`\geneuro`	`\eurobars`
`\EUR{`*bedrag*`}`	`\geneuronarrow`	`\eurobarsnarrow`
`\officialeuro`	`\geneurowide`	`\eurobarswide`

Belangrijkste macro is `\EUR` die gebruikt wordt om een bedrag te typesetten. Deze macro zorgt ervoor dat er een nette afstand tussen het bedrag en het eurosymbool wordt geplaatst. Bij het gebruik van de macro `EUR` speelt de optie `left` of `right` een rol.

Deze bepaalt of het euro symbool links –jawel!– danwel rechts van het bedrag verschijnt, afhankelijk van de conventies binnen de taal die je gebruikt. De macro `\euro` wordt door `\EUR` intern gebruikt om het symbool euro symbool op te roepen.

Default is dat `\officialeuro`.

Maar door `\euro` te herdefiniëren zijn er ook andere mogelijkheden. Zo biedt dit package bijvoorbeeld de mogelijkheid om met behulp van de hoofdletter C uit het geselecteerde font een euro te maken. Om dit voor elkaar te krijgen zijn de twee horizontale streepjes als een afzonderlijk glyph aanwezig.

De commando's `\geneuro...` doen de magie voor ons. De verschillende `geneuro` commandos leveren streepjes van verschillende breedte, zodat we het zelf gefabriceerde euro symbool er zo mooi mogelijk uit kunnen laten zien. En zo ziet het er tenslotte uit in een stukje tekst:

> Gaan we uit van een dergelijke indexatie en een verruiming van € 80.000 tot € 150.000, dan resteert er van de binnen te slepen € 850.000 nog een bedag van € 700.000.

En zo zien de 'gehackte' euro's eruit:
€ 1234 € 1234 € 1234
€ 1234 € 1234 € 1234

De package is fluitend te installeren. Alleen wel hier en daar wat slordig. Zo begint TEX te mekkeren dat je om eurosym gevraagd het terwijl het package eurosymbol levert. En bij printen moppert DVIPS over een checksum mismatch in de fonts... maar als we ons hier niets van aantrekken dan is het resultaat alleszins bevredigend.

MarvoSym package

Dit is een package dat geleverd wordt samen met een PostScript font. Het font is ontworpen door Martin Vogel, de package en het PostScript font zijn gemaakt door Thomas Henlich. Het is te vinden in:

`<CTAN>:fonts/psfonts/marvosym`

Installatie is ook betrekkelijk eenvoudig. Het PostScript font `marvosym.pfb` plaatsen we in een directory waar dvips het kan vinden

`texmf/fonts/type1/public/marvosym`

vervolgens voegen we de regel

```
fmvr8x Martin_Vogels_Symbole <marvosym.pfb
```

toe aan `psfonts.map` zodat dvips beseft dat het font aanwezig is. Deze regel zit als volgt in elkaar: `fmvr8x` is de naam van de font metrics file en wordt gebruikt door TEX, `Martin_Vogels_Symbole` is de naam van het font en `<marvosym.pfb` vertelt dvips in welke file het font te vinden is. Let goed op de hoofdletters anders komt het niet uit de printer.

Voor installatie van de fontmetrics file `fmvr8x.tfm` geldt dezelfde procedure als voor de voorgaande gevallen

`texmf/fonts/tfm/public/marvosy`

En tenslotte de moet de `marvosym.sty` file vindbaar voor TEX opgeborgen worden:

`texmf/TeX/Latex/contrib/misc`

Gebruik:

```
\usepackage{marvosym}
...
\EUR{} \EURtm \EURhv \EURcr
```

Het pakket biedt een officieel euro symbool (`\EUR`), een times (`\EURtm`), een helvetica (`\EURhv`) en een courier variant `\EURcr`.

En zo zien het eruit.

> Gaan we uit van een dergelijke indexatie en een verruiming van € 80.000 tot € 150.000, dan resteert er van de binnen te slepen € 850.000 nog een bedag van € 700.000.

En de andere varianten:
€ 1234 **€ 1234** € 1234 € 1234

Verder bevat het font nog een weelde aan heerlijke andere symbolen die je normaal gesproken niet gebruikt, o.a. tekens van de dierenriem.

Het Adobe eurofont

Ook dit is een PostScript (wat een verassing!) Het bevat drie font families (sans, mono, en serif), met de vier gebruikelijke stijlen (regular, italic, bold, bold italic). Elk font bevat één glyph en zit zo in elkaar dat elke charactercode een eurosymbool oplevert. Het font is te downloaden op de `ftp` site van adobe en er is zowel een macintosh en een windows versie.

```
ftp://ftp.adobe.com/pub/adobe/type
.../win/all/eurofont.exe
.../mac/all/eurofont.sea.hqx
```

De font metrics vinden we op CTAN

`<CTAN>:fonts/euro`

Macintosh gebruikers moeten er op letten dat de Adobe fonts in de mac versie andere namen hebben en dat de hier beschreven installatie daarom niet klakkeloos gevolgd kan worden.

De fonts hebben onmogelijke namen met underscores en nummers, brrrr...

installatie:
De font metrics:

`texmf/fonts/tfm/adobe/eurofont/`

Dan pakken we `eurofont.exe` uit, dan hebben we de PostScript fonts (`*.pfb` files), die plaatsen we in:

`texmf/fonts/tfm/adobe/eurofont/`

Vervolgens kopiëren we de inhoud van `psfonts.eur` (die we vinden in dezelfde directory als de `tfm` files) naar `psfonts.map`. Een kijkje in deze file leert dat alle regels dezelfde structuur hebben als de regel die we voor Marvosym hebben toegevoegd.

Macintosh gebruikers moeten handmatig de namen van de PostScript fonts in de file `psfonts.eur` aanpassen. Ik denk niet dat de gemiddelde Mac gebruiker, die gewend is aan plug and play dat een leuke procedure zal vinden...

Er worden alleen fonts geleverd en geen package om de fonts te gebruiken. Gebruik is mogelijk met plain TEX commandos. Dit is mijns inziens niet optimaal. Er zijn echter mensen die bezig zijn met packages voor dit font. Het beste is voor hen misschien om te wachten tot ze klaar zijn.

Als je ze toch wil gebruiken volgt hier een voorbeeld voor mensen die vergeten zijn hoe je ook al weer fonts gebruikt met plain TEX:

```
\font\eurosansr=_1 at 12pt
...
```

voor de andere fonts vul je op de plaats van `_1` de naam van de tfm file (zonder de toevoeging .tfm) en `eurosansr` vervang je natuurlijk door een andere naam. Nu kan je de fonts gebruiken met en commando zoals: `{\eurosansr C}`

De namen van de tfm files kun je vinden in de eerder genoemde file `psfonts.eur` (of natuurlijk gewoon in je directory).

Dit zijn de glyphs die je nu tot je beschikking hebt:

Euro Sans: € **€** *€* ***€***
Euro Monospace: € **€** *€* ***€***
Euro Serif: € **€** *€* ***€***

Let wel op dat de PostScript printer er niets van bakt als je werkt met *partial font downloading*. Dat wil zeggen, als je `dvips` alleen de 'nodige' fonts naar de printer laat sturen, dan denkt de printer dat er iets helemaal mis is. Let erop dat je als optie voor `dvips` de parameter `-j0` (minus, j, nul) meegeeft. Dan worden alle gebruikte `*.pfb` files naar de printer gestuurd en de printer kan daar goed mee uit de voeten.

Euro's money can buy

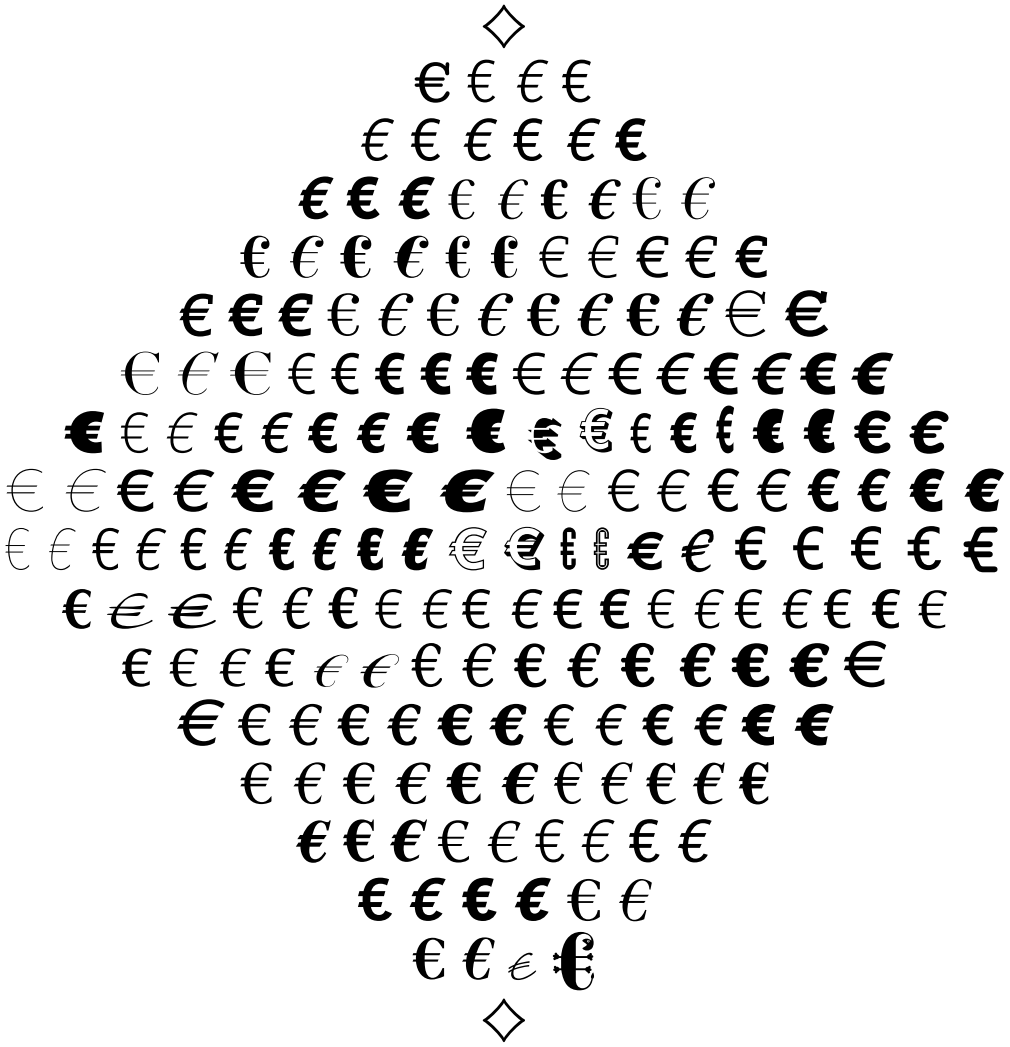
Bij de Fontshop kan een eurofont worden gekocht. Het betreft hier maar liefst 193 verschillende eurosymbolen. Een

verzameling euro's voor elke gelegenheid, meer euro's dan stropdassen, en euro's voor de man die alles al heeft. Er zit een handleiding bij maar daaraan heeft de TEX user niet veel.

Taco Hoekwater heeft echter gezorgd voor een implementatie van deze files, dus wie ze heeft gekocht kan op Taco's homepage de overige files ophalen.

Met `\usepackage{euro}` in de preamble ben je dan een bult euro's rijker, zoals hieronder te zien is.

Zo heeft de lezer meer dan 200 eurosymbolen binnen handbereik. Ooit zoveel dollartekens in uw ogen gehad?



fonts: background

A short introduction to font characteristics

Maarten Gelderman

abstract

Almost anyone who develops an interest in fonts is bound to be overwelmed by the bewildering variety of letterforms available. The number of fonts available from commercial suppliers like Adobe, URW, LinoType and others runs into the thousands. A recent catalog issued by FontShop [Truong et al., 1998] alone lists over 25.000 different varieties.[1] And somehow, although the differences of the individual letters are hardly noticable, each font has its own character, its own personality. Even the atmosphere elucided by a text set from Adobe Garamond is noticably different from the atmosphere of the same text set from Stempel Garamond. Although decisions about the usage of fonts, will always remain in the realm of esthetics, some knowledge about font characteristics may nevertheless help to create some order and to find out why certain design decisions just do not work. The main aim of this paper is to provide such background by describing the main aspects that might be used to describe a font.

keywords

Typefaces, design, font classification

The outline of the remainder of this paper is as follows. First I will discuss some basic font characteristics. Next some elementary, numerical dimensions along which properties of a typeface design can be asssessed will be discussed. A next section elaborates on those measures and some additional aspect of 'contrast' will be discussed. The final two sections briefly present a font classification along the dimensions discussed in the previous section and some implications.

Some elementary differences

Proportional and monospaced. A first difference between typeface designs that can be recognized is the spacing of fonts. Monospaced or typewriter fonts in which each character occupies the same amount of space can be distinguished from proportionally spaced fonts.

Computer Modern typewriter (monospaced): Winmvw

Computer Modern Concrete (proportionally spaced): Winmvw

Hardly anyone will dispute the statement that proporionally spaced fonts are more beautiful and legible than monospaced designs. In a monospaced design the letter i takes as much space as a letter m or W. Consequently, some characters look simply too compressed, whereas around others too much white space is found. Monospaced fonts are simply not suited for body text. Only in situations where it is important that all characters are of equal width, e.g., in listings of computer programs, where it may be important that each individual character can be discerned and where the layout of the program may depend on using monospaced fonts, can the usage of a monospaced font be defended. In most other situations, they should simply be avoided.

Romans, italics and slant A second typeface characteristic that will hardly be new for any TEX-user is the difference between Italic, Oblique (slanted) and Roman fonts. The difference between Italic fonts and the Roman fonts lies in their history. Italic fonts are the descendants of handwritten letter shapes, whereas the Roman fonts were originally chiselled in stone. Consequently, the romans look more rigid; the italics on the contrary show more elegance and are more 'curvy'. Furthermore, the shapes of some individual characters differ; this difference is most apparent when we look at α, g and a, g (here in the Italic and Roman variant respectively). The origins of the italics being in handwriting, they are usually slanted, whereas the romans are typically typeset upright. This, however, is not strictly necessary. Italics can theoretically be typeset upright and Romans may be slanted:

An upright Italic and a *slanted or oblique Italic*

An upright roman and a *slanted or oblique Roman*

Generally designers agree that text set in Roman is more legible than text set in Italic, although the readability of Italics accompanying different fonts may differ considerably,

1. This enormous variety is partially made possible by the introduction of electronic typefaces, which allow for worldwide distribution without exceptional cost. In 1950, that is before the advent of electronic typesetting Groenendaal could still attempt to list *all* typefaces readily available to an ordinary typesetter.

which is important if large pieces of text are typeset in Italics. Compare for instance:

A block of text set from Utopia Italics. Generally designers agree that text set in Roman is more legible than text set in Italic, although the readability of Italics accompagnying different fonts may differ considerably, which is important if large pieces of text are typeset in Italics.

A block of text set from Computer Modern Italics. Generally designers agree that text set in Roman is more legible than text set in Italic, although the readability of Italics accompagnying different fonts may differ considerably, which is important if large pieces of text are typeset in Italics.

If multiple slanted fonts are used in one piece of running text, it is important to ensure that the angle of slant is comparable, otherwise a page will look rather uneven.

Serif and sans serif. An issue that raised much discussion in the first half of this century (see e.g., Tschichold [1991]) but on which a communis opinio now seems to have been reached is the usage of serifed or sans serif fonts:

Computer Modern (with serifs)

Computer Modern sans (sans serif)

Whereas at the beginning of this century a large group of designers were of the opinion that sans serif designs were to be preferred as they were more modern, emphasizing the pure shape of the individual characters and omitting superfluous elements, it is now generally recognized that the serifs have an important function for the following, not always independent, aspects of legibility:

- Serifs make individual characters more distinct. In their sans serif variant many characters look remarkably, if not exactly, like mirror images of eachother. During the reading process they are easily confused, especially by persons suffering from dyslexia. The advantage of serifed typefaces over their non serif counter parts, in this respect, is easily seen from the following example:

  b d b d
  p q p q

- Serifs emphasize the begin and end of individual characters, compare e.g., rn with rn.
- Serifs emphasize the shape of words. It is generally recognized that experienced readers do not read individual characters, but read words and mainly use the upper half of a line of text for this purpose. The general claim is that the serifs facilitate this process. Just check it for yourself by looking of the next set of lines:

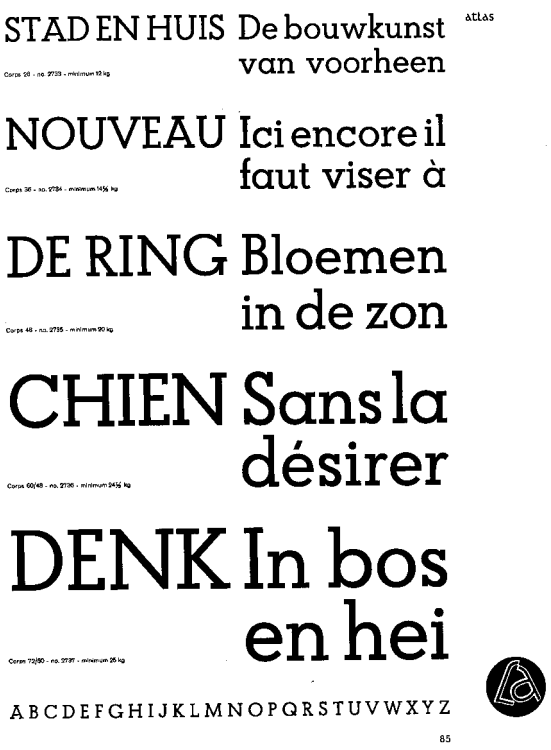


Figure 2. Font specimen of 'Atlas' (source: N.V. Lettergieterij Amsterdam [Undated]).

Now you miss the upper half of this line
This is a text: quer auer galapagos
This is a text: quer auer galapagos

Furthermore, serifs have an important function in shaping the personality of a type design. Different serifs—a set of possible serifs is given in Figure 1—give a typeface design a clearly distinct personality.

The first serif actually is no serif at all. The second one, the slab serif is orthogonal to the stem to which it is attached and has about the same width as this stem. Slab serifs are generally, but not necessary (lucida typewriter is a well-known example), used for monospaced fonts like Courier and Computer Modern Typewriter. Some proportionally spaced fonts, like the Computer Modern Concrete we encountered earlier in this paper, also have slab serifs. Those fonts are generally called Egyptiennes and are normally used for two purposes: display text in advertising and for typesetting labels on maps. A well known example is the Atlas, by the Amsterdam Typefoundry (see Figure 2). An important reason for using slab serifs in this latter type of copy may well be that the serifs clearly belong to the

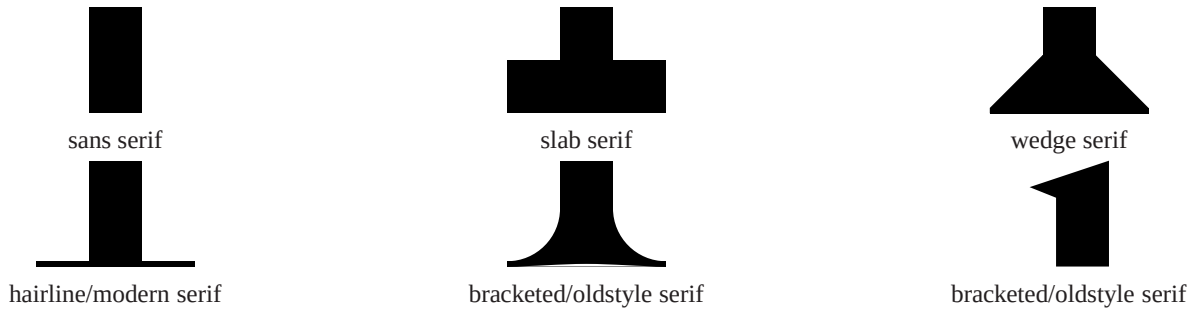


Figure 1. Different types of serifs.

letters and consequently are not likely to be confused with other elements on the map.[2]

The next type of serif, the wedge serif, has been popular in advertising and for book covers during the fifties and sixties of this century, but is hardly used nowadays. The main, and probably only, advantage of this design is that is is easily drawn by hand and still looks somewhat unusual.

The hairline or modern serif is typical of 'modern' typefaces like Didot or Bodoni (see Figure 3). Such serifs became popular in the second half of the eighteenth century. Great craftmanship was required to make the matrices needed to cast letters with those extremely thin serifs. Furthermore, great care should be taken during printing, as the hairline serifs were very fragile and could easily break. Nowadays, one does sometimes wonder whether those designs are the equivalent of Paganinis capricios for violin, is their main purpose not to show craftsmanshap rather than beauty? Nevertheless, one has to admit that a book in Bodoni, carefully typeset on the right kind of paper still looks stunning (apart form blackletter, Bodoni is one of the very few typefaces that looks good in combination with high contrast illustrations like woodcuts Groenendaal [1950]).

The serif we encounter most often, is the bracketed or oldstyle serif (both the lower and upper serif are shown in Figure 1). This is the traditional serif, found in fonts like Garamond, Bembo and Times.[3]

The dimensions of a typeface design

Size and design size The best known, and probably least usefull dimension of a font is its 'size'. Everyone has encounterd remarks like 'this text is set from a 10-points Bembo' and 'papers should be submitted in 12-points Times Roman'. Traditionally the size of a font is the height of the piece of lead from which the text is set. Nowadays the size of a font can generally be considered an almost useless figure. In most fonts it is equal to the height of the parentheses ('()'), but even that is not always the case. In wordprocessors, the point size will generally be equal to the distance between lines of text if you set linespacing to one. For practical purposes this knowledge is limited, the only thing about font size that is important is that most fonts have a design size. This is the size at which the font will look best. Although, using modern typesetting software like TEX, or any Windows or MacIntosh program it is generally possible to scale a font to any desired size, you will generally get better results if you stick to a size in the neighbourhood of the design size. For some popular fonts, like Times Roman or our good old Computer Modern, different design sizes even are available. This allows the careful designer to use all fonts at their optimal sizes. When using Computer Modern, the standard LATEX document classes even take care of this automatically: the footnotes, for instance, are set from a font with another design size than the font used for the main text. This ensures an equal level of 'grayness' accross the page and increases legibility (characters of fonts with a smaller design size are generally somewhat wider and heavier), look for instance at the difference between the next two examples:

Computer Modern with 5 point design size

Computer Modern with 17 point design size

The x-height A more important characteristic for practical purposes is the x-height of a font, which is exactly what the name implies the height of an x (or any other letter without ascenders or descenders) in the given font.[4] The x-height of a font, essentially determines the size of the font as it will be perceived by the reader. Fonts with an identical

2. A second reason for the preference for Egyptiennes and sans serif fonts in applications like map printing is that the contrast of those fonts typically is near unity, see the discussion on contrast later in this paper.

3. Times is somewhat peculiar in this respect: the bold characters use modern serifs, the ordinary romans oldstyle serifs.

4. The x-height of a font is readily available in TEX. If you want to specify a length in terms of the x-height of the current font, just use the measure ex, instead of a more traditional measure like cm or pt.

size, may have x-heights that differ surprisingly. The next two examples show Utopia and Garamond at the same size. The x-heights, and consequently the perceived size of the font, however, differ considerably:

Hamburgefont Hamburgefont

When combining fonts in running text, for instance when using typewriter or sans serif fonts in combination with an ordinary serifed roman, it is important to ensure that the x-heights of all fonts used are identical. A traditional problematic combination are the standard PostScript fonts Times, Helvetica and Courier. Thos fonts have quite different x-heights which distorts the evenness of a page if no measures are taken:[5]

Times Helvetica Courier

Fortunately, the new fonts selection scheme, as discussed by Siep Kroonenberg in another contribution to this issue, makes solving this problem rather easy: the default is to load each font at the same size; however, it is also possible to specify a scale factor in addition, which may be used to compensate for different x-heights.

Ascenders, descenders and capitals In addition to the x-height and font size, three other height-related dimensions of a font are available, the height of the capitals (e.g., K, H, and S), the height of the ascenders (e.g., k, l, and h), and the length of the descenders (e.g., j, g, and y). In many fonts the capital-height is equal to the height of the ascenders, sometimes, however the ascenders are slightly longer than the capitals. The main advantage of making the capitals slightly shorter than the descenders is that this gives a more even level of grayness accross the page, otherwise—especially when the ascenders are large relative to x-height—the capitals would stand out too much. An example of a font that uses slightly smaller capitals than ascenders is Garamond:

HhKkLlAk

The combination of x-height and ascender and descender heights roughly determine how economical a typeface is (Morison [1997] even claims that the general principle behind the evolution of font design is economy, and indeed more recently developed typefaces tend to be more economical than traditional ones), in other words: how many text can be put on a page without sacrificing legibility. Fonts with relatively large x-heights compared to their size can be used at small sizes. Consequently, they are rather economical: more lines of text can be put on a single page and more text will fit on a single line. However, the gain is not as large as one might hope for: fonts with relatively large x-height generally require some additional interline spacing.

Width and stem width Apart from the measures of font height, discussed in the previous paragraphs, we also need some measure of font width. TEX provides the user with an amount called em-space, the width of a single m, which for design considerations has relatively little importance. Somewhat more important is the average width of a font, generally measured [Rubenstein, 1988] by the total width of all lowercase characters. This width is also of importance when combining fonts. Although less perceptible than the x-height, fonts with different widths (given an identical height) tend to combine badly (this problem is mainly related to the 'rythm' of the font, to be discussed later in this paper).[6] Of course width also is related to the amount of text that can be put on a page; the larger the width the smaller the number of characters that fit on a single line. Not surprisingly, fonts with an x-height that is relatively large, tend to have a large width as well, thus reducing the economy gained by using such a font.

A final directly measurable characteristic of a font is stem-width: the width of the stems of letters like l. Of course this also influences the results when combining different fonts in a piece of text. The next example shows two monospaced fonts, along with a Times. With regard to stem width (and consequently blackness) Computer Modern typewriter combines far better with Times than the traditional Courier (but of course, the x-height still needs some adjustment).

Courier Times Computer Modern Typewriter

Some more complex dimensions

Color Although it is impossible to characterize a font completely by a set of numbers, we may refine the measurement presented till now to get some additional insight into the properties of a design. Most TEX-users, for instance, will have heard the remark that Computer Modern is 'too light'. This somewhat subjective criticism can be made more objective by calculating a measure of 'color'. This measure is defined as the ratio of the width of the set of all 26 lowercase letters, divided by the stem-width [Rubenstein, 1988]. In other words, color is a measure of the amount of paper left white: the higher the color-value of a font is, the lighter it looks. Color values for a number of popular fonts are provided in Table 1. It is evident that

5. The example also shows that color and rythm of the three typefaces differ.

6. Unfortunately TEX is only able to scale the height and width of a font simultaneously, so this problem is not easily solved. Future generations of TEX may well solve this problem.

Table 1. Color, weight and contrast of some popular fonts (the statistics for Times, Garamond, Helvetica, Bembo and Van Dijck are based on measurements presented in Rubenstein [1988], the statistics for both Computer Modern variants were kindly provided by Taco Hoekwater).

	color	*contrast*	*weight*
cmr12	197,111	1,703	0,146
cmr10	192,258	1,650	0,153
Times	156	2	0,17
Garamond	208	3	0,15
Helvetica	163	1	0,16
Bembo	184	2	0,16
Van Dijck	191	2,75	0,15

Times, which is the font of reference for most people, is a lot darker than the Computer Modern fonts. What also is noteworthy is that the 12 point Computer Modern is somewhat lighter that the 10 point variant. Finally, one may notice that, notwithstanding the common criticism that Computer Modern is 'too light' it is not the lightest font in the small set presented here: Garamond is even lighter. Apparently, color is not all there is to say. When we look at the other measures provided in this table, it seems as if Garamond is able to compensate for an apparent lack of color by a high contrast value.

Contrast Contrast, is defined as the ratio between the width of vertical and horizontal stems [Rubenstein, 1988]. Contrast is, roughly speaking, what makes a font lively, brilliant if you wish. If contrast gets extremely high, a font is hardly legible at all and only suited for use as a display typeface in for instance advertising. Similarly fonts with extremely low contrast are hardly legible. Endless discussions about optimal contrast values are, of course, possible, but there seems to be some general agreement that for, serifed typefaces, contrast should be somewhere between 2 and 3.5. It is evident from the data presented in Table 1 that Computer Modern scores rather low on the contrast (of if you wish, high in the 'dullness') dimension. The design simply lacks contrast to an extent that may impel legibility. The cautious reader may also have noticed the extremely low contrast value of Helvetica. Such contrast values are rather typical for sans serif typefaces, which tend to stress evenness, often at the cost of legibility.

There is another aspect of contrast that deserves attention: contrast also is an indication of the 'fragility' of a font. At low resolutions (or looked at from large distances) designs with high contrast may be seriously distorted. This is one of the main reasons why sans serifed typefaces (and typewriter and slab serif fonts, which also tend to have low contrast values) are the fonts of choice for transparencies, traffic signs and computer displays.

Weight A final, common dimension of a font is its weight. Color measures the darkness of a font as it appears to the reader who looks at a page of text. Weight is used to assess the darkness of the individual letters and it calculated by dividing the vertical stemwidth by the x-height of the font. According to Rubenstein [1988] if weight lies outside the range 0.15–0.2, legibility suffers. Apart from the 12 point Computer Modern all fonts presented in Table 1 are within this range. Times is the most 'weigthy' design in the set of fonts presented here.

Additional aspects of contrast

Contrast is one of the more important aspects of a type design. However, the measure of contrast presented above, does not cover this aspect completely. A first additional aspect of contrast is the axis of contrast, or the angle at which the broader parts of the characters appear. If we compare, for instance, the design of Bodoni (see Figure 3) with Bembo (see Figure 4), it is not only clear that contrast of Bodoni is higher than that of Bembo, but also that the axis of contrast differs. This is most easily seen, by comparing the o or the e of both fonts. In Bodoni, contrast is orthogonal to the baseline, whereas in Bembo, it is slanted to the left.[7] The axis of contrast has little influence on legibility of a typeface, although the axis of contrast is related to contrast and hence influences legibility indirectly.[8]

The second additional aspect of contrast, frequency, is a far more important determinant of legibility. Figure 5 show the sensitivity of the human eye as a function of frequency. Sensitivity is, roughly, defined as the ease with which for instance *individual* lines, drawn on a sheet of paper can be distinguished. If the lines are very far apart, that is frequency is low, the human eye is simply not able to focus on both lines simultaneously and sensitivity is low. If the lines are very close to eachother, frequency is high, the human eye does not distinguish individual lines any more. Although a page may contain black and white lines, it is perceived as being gray.[9] The ability of the human eye

7. If one mentally imagines the o begin drawn on paper with a broad brush or pencil, the brush would be hold horizontally when drawing the Bodoni o and at a 30° angle when drawing the Bembo o.

8. To maximize contrast, the horizontal parts have to be as thin as possible and this can only be accomplished using a 'horizontal brush'.

9. Frequency is not defined in terms of lines per inch but in terms of lines per degree of visual angle. If the sheet of paper is closer to our eyes, the number of lines per degree of visual angle diminishes, although the number of lines per inch remains the same. In this way

Figure 3. Font specimen of 'Bodoni' (source: Tschichold [1992]).

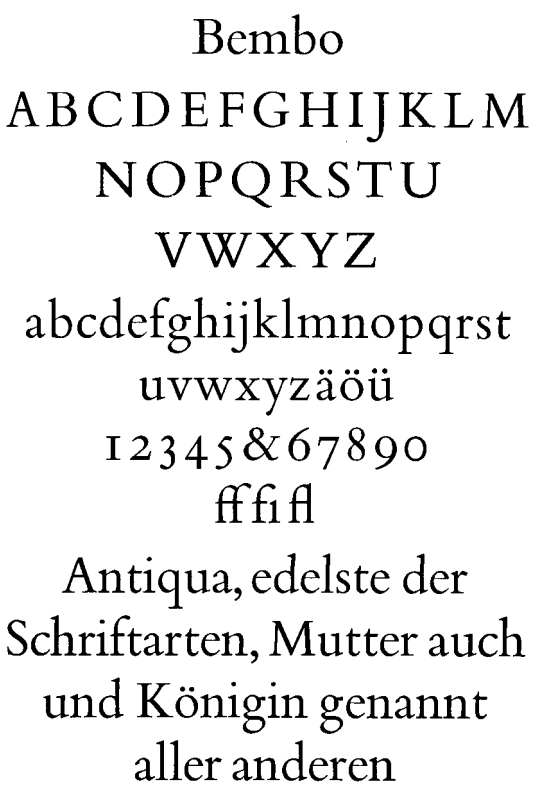


Figure 4. Font specimen of 'Bembo' (source: Tschichold [1992]).

to perceive individual lines, rather than no lines at all, or some level of gray, is at a maximum somewhere between 6 and 11 cycles per degree. Of course, in order for a typeface design to be legible, it is highly desirable that the individual strokes of the characters are easily discernible. Unfortunately letters do not consist of simple lines but are slightly more complex: a single number will not suffice to describe the frequency of a font. A number of frequencies will be present on a single page. Fortunately, using Fourier analysis it is possible to find those frequencies and make a plot of them, as is done in Figure 6 for three popular typeface designs: Times, Helvetica and Courier. Now we can look for a dominant frequency which hopefully lies some where between 6 and 11 cycles per degree. The results confirm our expectations: both Helvetica and Times show a clearly distinguishable peak in their frequency distribution at about the point of maximum discernability to the human eye. Helvetica, however, shows a second peak, which will make the design less readible. Courier, finally shows at least four peaks in its frequency distribution.

From characteristics to classification

The characteristics mentioned in the previous section, provide the clues that can be used to build a classification of typefaces. The traditional classification scheme distinguishes four categories of serifed typefaces: Venetian, oldstyle, transitional and modern. Venetian typefaces have been in use since about 1470. They are hardly distinguishable from oldstyle typefaces, which have been in use since about 1500. Both categories of fonts share a slanted axis of contrast and the usage of, not surprisingly, oldstyle serifs. Capitals, typically, are somewhat smaller than the ascenders, they end where the serifs of ascenders start. One reason for this is that the ascenders and descenders of those fonts are relatively long and their x-height is relatively small. Furthermore, those fonts are typically relatively light, and contrast is not extreme. To distinguish a

the individual lines that look like uniform gray at reading distance, become distinguishable at closer examination. At a reading distance of about 40 centimeter, frequency in lines per inch is about two times as high as frequency in lines per degree of visual angle.

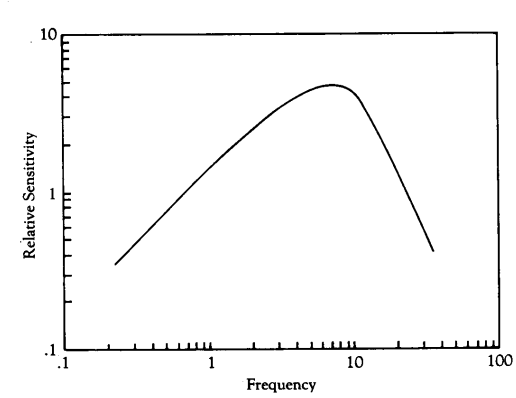


Figure 5. Sensitivity of the human eye as a function of frequency (in cycles per degree of visual angle) (source: Rubenstein [1988]).

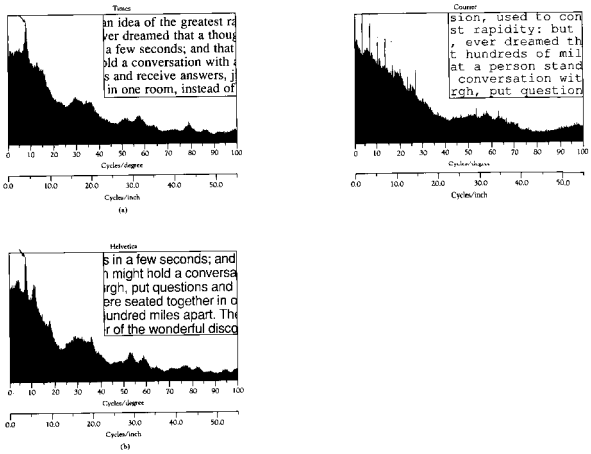


Figure 6. Results (power spectra) of Fourier analysis on text samples in three popular typefaces (source: Rubenstein [1988]).

Venetian from an oldstyle two features are of importance: first oldstyle fonts usually have a horizontal crossbar of the lowercase e, whereas this crossbar in a Venetian is at an angle of about 20° with the baseline (like in the 'Heineken' logo). Furthermore, the oldstyle capital M has the usual serifs, whereas the Venetian M has double serifs. Prime examples of oldstyle fonts are Garamond, Baskerville and Caslon. Popular Venetians are Cloister, Centaur and many of the designs by Goudy.

The first transitional font was the designed for French governmental publications in about 1702, but only came into general usage at about 1755. Although the serifs of those fonts are already horizontal, the contrast axis is not yet orthogonal to the baseline, but more upright than in the Venetian or oldstyle typefaces. It is generally claimed [Morison, 1997] that the ascenders are as high as the capitals in those transitional fonts, however, examination of some font specimens learns that this rule is not universally valid. Similarly, although the transitional fonts are supposed to have tablenumbers instead of oldstyle numbers,[10] this also is not always the case. The transitionals are generally blacker than oldstyle fonts; they look stronger, but less elegant.

Finally the moderns, of which Bodoni and Didot are the prime examples, can be found from 1790 on. The development of those typefaces continues the development started with the transitional fonts. The x-height slightly increases and the capitals are as high as (and sometimes even slightly higher than) the ascenders. The axis of contrast now is completely vertical and the serifs are horizontal. Contrast often is extreme, a page set from Bodoni looks brilliant. Although the page may look particularly well from a distance, legibility may suffer from this extreme contrast. Other moderns, like Egmont and Walbaum, are less extreme in this respect and consequently more legible. Tablenumbers are the rule, but exceptions may still occur.

Some implications

Typefaces, of course, neither were nor are designed with the classification or the numerous characteristics mentioned above in mind. The classification is not perfect, particularly recently developed font are difficult to classify. As a taxonomy, the classification scheme is useless, it merely functions as a starting point in determining the characteristics of a typeface, and the way it may be used. Typography remains an art, not a science, and each rule has its exception, but some rules of thumb may nevertheless help.

In the previous sections numerous aspects of font selection have already been mentioned. Monospaced fonts are generally not the best choice. Only for typesetting computer programs and similar applications, may they be the preferred kind of typeface. For applications like traffic signs, transparencies, computer applications and other messages that have to be read at low resolution or from a large distance, typefaces with low contrast, particularly sans serif and slab serif typefaces are generally preferred.

For typesetting large amounts of text, e.g., in a journal or a book, serifed typefaces are generally the best choice. If the result has to be striking modern typefaces are preferred. They may draw attention to a magazine the consumer otherwise wouldn't buy or to a feature article that otherwise might be skipped by most readers. Modern typefaces may

10. Tablenumbers all have the same size and do not have ascenders and descenders. Oldstylenumbers, on the contrary, differ in size and some numbers (e.g., 9) have descenders, whereas others (e.g., 6) have ascenders.

also be the font of choice because they blend well with illustrations or emphasize the 'designer-like' atmosphere of a book. Art books are a typical example.[11]

If it may be assumed beforehand that a text will be read, for instance in the case of a novel, oldstyle and transitional designs are preferred. Legibility of those designs is better than that of any other font category. Economy may be one of the criteria for font selection: with transitionals generally more text can be put on a given amount of paper than with the oldstyle fonts. Oldstyle fonts, on the other hand may be slightly more legible and, more importantly: they look more elegant. Selection of a particular typeface may also be guided by other considerations: Caslon is a fairly appropriate choice for a text by Spinoza, for a French novel from the early 19th century a Didot may be the right choice, just because of the contemporary atmosphere elucided by such a design.

After a certain typeface has been selected, some general guidelines may be drawn knowing its place in the classification scheme. Again, those guidelines are no laws, but mainly rules of thumb. With Venetians and oldstyles the œ and æ ligatures may be used, and usage of the fi, fl, and fli ligatures is almost required. When using a modern or transitional, the f-based ligatures can be missed, and usage of the other ligatures generally looks kind of overdone.

Font selection for the body text also has some implications for other design decisions. One of the charms of oldstyle fonts is that they look so quiet. To maintain this feature, section and paragraph headers may be typeset from an ordinary roman or from small capitals rather than the more commonly encountered boldface variant. In some cases, depending on how similar to the roman font this variant is, an italic may also work. Combined with modern faces, however, a design in which only ordinary roman and small capitals are used, looks just too withdrawn. The timidity of such a design just does not mix with the aggresiveness of a modern font.

A final remark, may be made about the combination of different typefaces in a design. Generally speaking this requires that both typefaces are clearly distinct. Furthermore it mosttimes works best when the typefaces used for headers and other sparingly used features is blacker than the font used for body text. Thus a Helvetica for section heading with a body text out of Times may wel work. Bembo for headings with Garamond for the body text (or vice versa) will just be plain ugly. Bodoni for the headings with a body out of Garamond may work (if used with care), Garamond for the headings with Bodoni for the body will probably be ugly, etc.

Of course, the rules mentioned above have their exceptions. The only way to find out what works is to experiment. The guidelines given may just help to reduce the number of options to be investigated and to explain afterwards what did and didn't work. And this feature, combined with an urge to communicate the joy playing around with fonts gives me, was the main aim I had with this article. To anyone who wishes to pursue the topics touched upon in this paper in more depth, I can recommend reading Tschichold's treasury of art and lettering. For those interested in technical details, Rubenstein's monograph is a valuable source book.

References

M. H. Groenendaal. *Drukletters: hun ontstaan en hun gebruik*. De technische uitgeverij H. Stam, 1950.

Stanley Morison. *Letter forms: typographic and scriptorial*. Hartley & Marks, 1997.

Richard Rubenstein. *Digital typography: an introduction to type and composition for computer system design*. Addison-Wesley, 1988.

N.V. Lettergieterij Amsterdam. Selección de tipos modernos, Undated.

Main-Linh Thi Truong, Jürgen Siebert, and Erik Spiekermann, editors. *Digital Typeface Compendium: Font Book*. FontShop International, Berlin, 1998.

Jan Tschichold. *Schriften 1925–1974*. Brinkmann & Bose, berlin, 1991.

Jan Tschichold. *Treasury of alphabets and lettering*. Lund Humphries, London, 1992.

11. The majority of the applications in which modern typefaces can be used share another characteristic: they are typically printed on glossy paper which not only combines well with the atmosphere of e.g, a Bodoni, but also is a prerequisite for adequate printing of the extremely thin hairlines of this typeface.

development

Introducing Eetex

Taco Hoekwater
Hans Hagen
NTG TEX Future Group
ntg-toekomsttex@ntg.nl

abstract

This article gives an introduction to eetex. Eetex is an extension to e-tex 2.0 that defines a collection of new primitives. Most of these deal with list data structures, but some other things are added as well.

keywords
e-tex, eetex, extensions, SGML

Where it comes from

Eetex is the NTG TEX Future Group's *toy program* to investigate new extension proposals for E-TEX and NTS. Most of the web programming for eetex is done by Taco Hoekwater, and most of the prototyping and debugging is done by Hans Hagen.

Because of this, eetex currently contains mostly things that are desired by one of us two:

- Hans's desire to simplify and speed-up CONTEXT;
- Taco's desire to parse XML and SGML.

The current program's code is *neither stable nor bug-free!*. Use this program at your own risk. Eetex is a moving target, and the implementors do not care much for backward compatibility. We do not even guarantee that the current added primitives will still exist tomorrow. We have probably made many mistakes and omissions in this version that are so serious that we may have to re-think the entire approach, and eetex might change drastically because of that.

Eetex does not have any associated logo, it's name is precisely as given here, 5 lowercase letters with normal capitalization when that is grammatically normal for the language in which the text is written.

Notes on using eetex

Eetex writes it's format files with extension `.eefm` (`.eef` on real-mode MSDOS) to distinguish itself from other executables that use the same TEX source for their format files.

Starting eetex and generating formats for eetex is a lot like using E-TEX in extended mode. For example,

```
eetex -ini *plain
```

creates a new format file for `plain.tex` with the E-TEX extensions and the eetex extensions both enabled. Don't forget the `*` in the command line, or you will end up with a halfway solution.

New primitives for list manipulations

All following primitives deal with lists. Lists behave a lot like normal TEX macros, but they have an internal substructure that can most easily be thought of as specifying separate token list items in a list, separated by one or more tokens that are handled specially.

The primitives listed below are really quite primitive. Higher-level macros have to be written to make real use of the new functionality. Depending on the design of these macros, lists can work either as arrays or as lists or as queues or as unique sets.

Without going into very much detail, here are some of the problems that can benefit from the addition of lists to TEX's repertoire of basic data structures:

- Parsing input data.
  Some TEX macro packages read a lot of plain ASCII data that has to be split into separate tokens for processing. An example would be a plotting package, another example would be a macro package like CONTEXT's `supp-ver`, that interprets verbatim text.
  The macros that do this usually process the argument one character at a time using a brute force approach. In most of these cases, the same work can be done a lot easier and faster using lists.
- Parsing key=value pairs.
  There are commands available to get just a portion of an item. This comes in handy when macros are used to parse things like

  ```
  \includegraphics[height=6cm]{figure.eps}
  ```

- Creating cross-reference lists.
  Labels are usually required to be unique withing a certain scope. Lists make it fairly simple to create an application wherein the scope of the uniqueness (document, chapter, section) can be changed.

Lists also make it easier to write macros that differentiate between different types of labels (sections, formulas, tables).

- Economizing the hash table.
  TEX's hash table usually has a limited size (even in 'dynamic' versions like web2c, there still is an upper limit). But there usually is a lot of memory for token lists available. Therefore, it makes sense to store commands as token lists.

  Every list occupies just one hash entry, regardless of it's size. Yet, a list can be used to save the value of a macro in it's items.
- Database publishing.
  Lists can be used for publication of for instance production or bibliographical databases, because lists allows you to use more than 50.000 cross-refs within on document easily using the tricks from the previous two items.
- Exporting information from within a group.
  Lists make it possible to save values that are computed within a level of TEX grouping without having to resort to `\global` definitions.
- Maintaining information stacks.

Tracking what is going on

`\tracinglists`
Just like all the other tracing commands, this primitive displays information about what the executable is doing. Currently, this is somewhat more like debugging information than something a user might be interested in. Usable values are 1 and 2.

Related to this new primitive, setting `\tracingstats` to 3 or 4 results in a large amount of memory allocation information.

Splitting arguments into items

`\listsep`
This is a new internal token register, whose contents is used both as item separator when the user specifies a list and as filler between items when an expansion of a list takes place. See below for an example of the usage of `\listsep`.

The logic by which the separation happens is a perhaps little strange, but the current solution turned out to be the most desirable behaviour. It goes like this:

- The first token from the expansion of `\listsep` is used as separator token in list specifications. This token (on its own) is used to decide where items end and a new one starts.
- However, all subsequent tokens from `\listsep` are removed from the input as well, provided that they appear directly after the item separator and in the correct relative order.
- If `\listsep`'s current value is empty, the input will be split up into separate tokens, each item consisting of precisely one token from the input.
- Initex initializes `\listsep` to the equivalent of `\listsep={,}`.

The trick with subsequent tokens makes it possible to say for example `\listsep{, }`, making certain that there are no items in the resulting list that start off with a space (as might be the case for `\listsep{,}`).

How to define a list

There are many ways to define a list or change the definition of an already existing list. The following new primitives work directly on lists.

```
\listdef <csname> [{item text}|<listcs>]
\appdef   "
\predef   "
\insdef   "
\elistdef "
\eappdef  "
\epredef  "
\einsdef  "
```

The four primitives that start with `"e"` expand the `item text`, the others don't. This difference is anologous to the difference between `\def` and `\edef`.

All eight primitives are assignments, just like the 'normal' `\def`. It is much easier to give an example of how to use these commands than it is to try to explain the formal logic, so here is an example of the four different types of definitions:

```
\listsep{, }
\listdef \mylist {noot,mies}
\appdef  \mylist {wim}
\listdef \first  {aap,}
\predef  \mylist \first
\insdef  \mylist {noot,zus}
\listsep{ }
\message {\mylist}
\bye
```

The terminal output of this example is:

```
zus aap  noot mies wim
```

with two spaces between `aap` and `noot`.

Wat happens on the preceding lines is the following:

1. Initializes `\listsep`
2. This line defines the csname `\mylist` to be a 2-item

list consisting of the items noot and mies.
3. This appends the item wim. The list \mylist now has 3 items.
4. This defines \first as another 2-item list. The second item of this list is empty.
5. Prepends that new list to \mylist. (\mylist now is "aap" "" "noot" "mies" "wim")
6. Requests insertion of the items noot and zus. noot already exists in the list, so that one is ignored and zus is added.
7. Changes \listsep for the subsequent expansion.
8. Expands \mylist. Notice that you get 2 spaces around the empty item.

All of these primitives adhere to standard TEX grouping, and they all understand the \global prefix. The control sequence name that becomes defined is always considered to be \long and never considered to be \outer.

An completely empty item text does nothing if it is used together with the insertion or addition primitives, but \listdef \mylist {} *does* change \mylist into a csname that expands into a list (of zero items). Empty lists as well as empty items are legal (both have their uses).

Lists expand into a token list that is a concatenation of the items' contents, with the items separated by the current expansion of \listsep. This expansion happens without the need for the user to do anything; lists are 'callable' just like macros even if their interal structure is quite different.

But lists expand into the internal representation of a number if TEX is looking for an integer (for primitives like \ifcase, \number, counter assignments, etc.) The returned number is the number of items in the list. This gives you a simple way to measure the length of a list.

\newlist<csname> <number>
Creates a list with <number> amount of items. This is useful for array specifications, and for extending or shortening an already existing list. If you are extending an already existing list or if you are creating an entiry new one, all new items in it will be empty.

Already existing items keep their value. If you are shortening an existing list, the items that are cut off are irretrievably lost though. Subsequently extending the list will *not* give them back.

Mapping macros to lists

\scanlist<listcs> <token>
Explicitly expands the list that is pointed to by <listcs>. But instead of inserting the current meaning of \listsep, it inserts the <token> between every item and before the very first item of the list, and it adds braces around the separate items. The idea is (see below for an explanation of \quitlist):

```
\def\noot{noot}
\def \test#1{\def \tempa{#1}
      \ifx \tempa \noot
        \message{done}%
        \quitlist 1
      \else
  \message{#1}%
      \fi }
\scanlist \mylist \test
```

If \mylist consists of the three items aap, noot and mies, the \scanlist expands into

```
\test{aap}\test{noot}\test{mies}
```

\quitlist<number>
Quits from the <number>-ed input level above the current one that is a token list which is the result of expanding the \scanlist primitive. This sounds complicated, but it simply means that

```
\quitlist 1
```

could be used in the \test macro above to escape out of the list's expansion once the condition was met (so that \test{mies} was never expanded). On long lists, this can safe a lot of processing time. In nested definitions, numbers higher than one might also be useful.

The command \quitlist 0 is a special case: it kills the current token list, regardless of its type. This is likely to be the expansion of a macro, and it means that macros can actually quit themselves (like exit and return do in other programming languages).

Finding out if this is a list

\iflist<csname>
Returns true if the <csname> represents a list.

Besides this, \ifx returns true if it's two csnames are two lists which have the same number of items *and* whose expansions fully agree. Note that the comparison is expansion-based, such that in the following example:

```
\listdef\mylist{no,ta}
\listdef\mylisttwo{n,ota}
\listsep{}
\ifx\mylist\mylisttwo
```

the \ifx evaluates to true.

\ifx returns false in all other cases, including the comparison between a one-item list and a macro that has precisely the same expansion.

\ifcase, \ifnum and \number return the number of items (which would be 2 in this case).

Searching for (part of) an item

```
\ifhasitem <listcs> [{item}|<listcs>]
\ifsubitem <listcs> [{subitem}|<listcs>]
\ifsublist <listcs> [{items}|<listcs>]
\ifsubset  <listcs> [{items}|<listcs>]
```

These are four new \if tests.

\ifhasitem tests for the existance of one item. You can specify more than one item in the items part if you want, but those are never looked at.

\ifsubitem tests for an item that starts with subitem. This is especially useful for key=value pairs.

\ifsublist tests for items appearing in the specified relative order. Intervening items are allowed, but the relative order must be maintained.

\ifsubset tests for all the items appearing in any order at all. There is currently no way to test whether items appear more than once in the list.

```
\itemnumber <listcs> [{item}|<listcs>]
\subitemnumber <listcs> [{subitem}|<listcs>]
```

These are two new expandable primitives that return TEX internal numbers. Here it is also possible to specify extra items if you want to, but they are ignored completely. If the requested (sub)-item does not exist, these commands return zero.

The above six new primitives have an extra sideeffect: when the tests are succesfull (either the \ifs are true or for \...number there is indeed such an item), the requested item's info is saved in two global variables that can be queried by the user. These are:

```
\lastitemnumber   % a counter
\lastitemdata     % a \long macro
```

If the test is unsuccessful, \lastitemnumber will be 0 and \lastitemdata will be empty.

If the test is successful, \lastitemnumber will be the itemnumber of the requested (sub)item and \lastitemdata will be the text of that item.

If the request was for a subitem, \lastitemdata contains *only* the trailing contents of the item, with one optional layer of containing braces stripped.

The main advantage of this side-effect is that it allows you to replace the construction

```
\ifsubitem \mylist {clip}
 \getitem \mylist \subitemnumber{clip} to \tempa
 \EA\def\EA\keyval\EA{\EA\stripeq\tempa=}\fi
```

with this code, which is both faster and a lot cleaner to look at:

```
\ifsubitem \mylist {clip=}
 \EA\def\EA\keyval\EA{\lastitemdata}%
\fi
```

Note: for \ifsublist and \ifsubset, the values will be the info of the last specified item.
Second note: \lastitemnumber might be zero even within the true branch (this is the result you get from checking for the existence of an empty item).
One last note: \lastitemdata is a \long macro after the first use of one of these six primitives, but for initex it is initialized to a weird typeless primitive. The contents or both csnames do not survive dumping and undumping formats, so they can not be used in \everyjob.

Manipulating items

```
\getitem <listcs> <number> to <csname>
\setitem "
\delitem "
\insitem "
```

Four new primitives to play with separate items.

\getitem defines <csname> to be the meaning of the relevant item of the list. \setitem works the other way around. \delitem is like \getitem but it also destroys the item of the list (the list actually becomes shorter), \insitem is \delitem inverted (the list gets longer).

Negative values for <number> count from the tail forward, such that -1 means the last item, and 1 the first item. These primitives are quite flexible, but <csname> has to be a macro. Using other primitives like \message will give you an error message.

Argument specifications (both # marks and delimited text) for macros are saved into the list as well, so that it is possible to do this:

```
\def\tempa#1#2{\message{(#1, #2)}}
\setitem \mylist 1 to \tempa
\getitem \mylist 1 to \tempb
\tempb {A}{B}
```

Other primitives and functionality

The primitives below have nothing to do with lists, but are added because we thought they might be useful.

\eeTeXversion
The first of these is for maintenance reasons only. \eeTeXversion is a read-only register that gives you the release number of the version of eetex that you are using. Currently, it returns the value 2.

Toks manipulation

The commands `\apptoks` and `\pretoks` allow you to append or prepend tokens to a token register. Here are four simple examples:

```
\apptoks \everyjob ={\message{This is eetex}}
\pretoks \output   ={\message{Output called}}
\apptoks \toks5    \toks2
\pretoks \mytoks   \toks0
```

`\sgmlmode`

This is a read-write register with default value 0. Setting this register to a non-zero value changes the way TEX reads control sequence names. With `\sgmlmode=1`, a csname ends at *either* the next space *or* at the complement character of it's escape char. A complement character is not removed from the input, and it is also not part of the csname (it is usually the first character of the argument specification). A trailing space is removed though, as in TEX.

The following complements are currently defined, while waiting for the implementation of a more general solution that is already proposed by Michael Vulis: `<↔>`, `(↔)`, `[↔]` and `&↔;`. This set is designed such that parsing XML-like syntax is rather simple.

Suppose you want to parse the following input:

```
<p indent=none>First paragraph</p>
<p>Second paragraph</p>
<s3 align=center>A subsubsection
```

The `p`'s are simple to do in current TEX, using a definition like:

```
\def\p#1>{....}
\catcode `\< = 0
```

where in the second paragraph macro the argument `#1` is empty.

But the `s3` is a little trickier. If there are `s3`'s, it's a safe bet that there are also `s1` and `s2`'s. The logical thing would be to define `\s` in such a way that it looks ahead to see what the next character is and then do an `\ifcase` based on the result. This is a little cumbersome, but quite straightforward.

Unfortunately, there are also `</p>` constructions. Therefore, you also need to define `\/` to do such a lookahead. Besides the fact that `\/` has a primitive meaning (italic correction), it also has to do a rather huge switch matching following tokens to predefined 'element' names.

This is why `\sgmlmode` was invented. The following definitions do the same trick (almost) without all that fuzz:

```
\def\sdef#1{\expandafter\def
 \csname #1\endcsname##1>}

\sdef{p}{...}
\sdef{/p}{...}
\sdef{h3}{...}
\sdef{/h3}{...}
\catcode`\< = 0
\sgmlmode=1
```

The macro `\sdef` creates a `\csname` construction that precisely matches the command that is needed for one specific case; once eetex starts parsing the example text input, it will end every `\csname` (all of which start off with a `"<"`) only when it reaches the next `">"` *or* when it reaches the next space, which gives the desired result.

Watch out for the fact that once you get into `\sgmlmode`, spaces at the end of control sequences become required. Exchanging the last two lines from the example would not have worked:

```
\sgmlmode=1
\catcode`\< = 0
```

creates a total of 4 tokens for the 2nd line: `"\catcode`\<"`, which will probably result in an 'undefined csname' error, and the three commands `"="`, `" "` and `"0"` that will be typeset!

New dimension specifiers

Eetex recognizes two new types of dimensions: `px` and `%`.

A `px` corresponds to a pixel, using a resolution of 96 dots per inch to calculate the conversion factor: `96px` equals `1in`.

The `%` is introduced because certain kinds of input use it to signify a percentage of a default or previous value. Setting up `%` in a meaningful turned out ot be quite tricky, so the current implementation maps one on one to `sp`: `100%` equals `100sp`. Of course, `%` only works if the `\catcode` of `%` is 12.

Where to get it

Eetex is available as pre-compiled binaries from the following URL:

`http://www.cybercomm.nl/~bittext/eetex/`

There are binaries for DOS/Win95 using the DJGPP compiler and Linux binaries using glibc. Both of these are linked use web2c version 7.2. In the future, there may be a source distribution (web change files) available as well.

The NTG TEX Future Group can be reached at `ntg-toekomsttex@ntg.nl` (our mailing list). subscription requests can be sent to `majordomo@ntg.nl` with body "subscribe ntg-toekomsttex [e-mail address]".

Hàn Thê Thành,
Sebastian Rahtz,
Hans Hagen

pdftex
The pdfTEX users manual

1 Introduction

The main purpose of the PDFTEX project was to create an extension of TEX that can create PDF directly from TEX source files and improve/enhance the result of TEX typesetting with the help of PDF. When PDF output is not selected, PDFTEX produces normal DVI output, otherwise it produces PDF output that looks identical to the DVI output. The next stage of the project, apart from fixing any errors in the program, is to investigate alternative justification algorithms, possibly making use of multiple master fonts.

PDFTEX is based on the original TEX sources and WEB2C, and has been successfully compiled on UNIX, AMIGA, WIN32 and MSDOS systems. It is still under beta development and all features are liable to change. Despite its β–state, PDFTEX produces excellent PDF code.

As PDFTEX evolves, this manual will evolve and more background information will be added. Please be patient with the authors.

2 About PDF

The cover of this manual shows a simple PDF file[1]. Unless compression and/or encryption is applied, such a file is rather verbose and readable. The first line specifies the version used; currently PDFTEX produces level 1.2 output. Viewers are supposed to silently skip over all elements they cannot handle.

A PDF file consist of objects. These objects can be recognized by their number and keywords:

```
8 0 obj << /Type /Catalog /Pages 6 0 R >> endobj
```

Here `8 0 obj ... endobj` is the object capsule. The first number is the object number. Later we will see that PDFTEX gives access to this number. One can for instance create an object by using `\pdfobj` after which `\pdflastobj` returns the number. So

```
\pdfobj{/Type /Catalog /Pages 6 0 R}
```

inserts an object into the file, while `\pdflastobj` returns the number PDFTEX assigned to this object. The sequence `6 0 R` is an object reference, a pointer to another object. The second number (here a zero) is currently not used in PDFTEX; it is the version number of the object. It is for instance used by PDF editors, when they replace objects by new ones.

In general this rather direct way of pushing objects in the files is rather useless and only makes sense when implementing for instance fill–in field support or annotation content reuse. We will come to that later. Unless such direct objects are part of something larger, they will end up as isolated entities, not doing any harm but not doing any good either.

When a viewer opens a PDF file, it first goes to the end of the file. There it finds the keyword `startxref`, the signal where to look for the so called ‘object cross reference table’. This table provides fast access to the objects that make up the file. The actual starting point of the file is defined after the `trailer`. The `/Root` entry points to the

1 See figure 1

```
%PDF-1.2
3 0 obj <<
/Length 4 0 R
>>
stream
1 0 0 1 91.925 759.924 cm
BT
/F51 9.963 Tf 0 0 Td[(W)80(elcome)-250(to)
-250(pdfT)]TJ 67.818 -2.241 Td[(E)]TJ 4.842
2.241 Td[(X!)]TJ 138.923 -654.744 Td[(1)]TJ
ET
endstream
endobj
4 0 obj
162
endobj
1 0 obj <<
/Font << /F51 5 0 R >>
/ProcSet [/PDF /Text]
>> endobj
2 0 obj <<
/Type /Page
/Contents 3 0 R
/Resources 1 0 R
/MediaBox [0 0 595.273 841.887]
/Parent 6 0 R
>> endobj
7 0 obj <<
/Type /Encoding
/Differences [ 0/.nothdef 5/dotaccent
/hungarumlaut/ogonek 8/.notdef 9/fraction
10/.notdef 11/ff/fi/fl/ffi/ffl/dotlessi
/dotlessj/grave/acute/caron/breve/macron
/ring/cedilla/germandbls/ae/oe/oslash/AE
/OE/Oslash/space/exclam/quotedbl/numbersign
/dollar/percent/ampersand/quoteright
/parenleft/parenright/asterisk/plus/comma
/hyphen/period/slash/zero/one/two/three/four
/five/six/seven/eight/nine/colon/semicolon
/less/equal/greater/question/at/A/B/C/D/E/F
/G/H/I/J/K/L/M/N/O/P/Q/R/S/T/U/V/W/X/Y/Z
/bracketleft/backslash/bracketright
/circumflex/underscore/quoteleft/a/b/c/d/e
/f/g/h/i/j/k/l/m/n/o/p/q/r/s/t/u/v/w/x/y/z
/braceleft/bar/braceright/tilde/dieresis
/Lslash/quotesingle/quotesinglbase/florin
/quotedblbase/ellipsis/dagger/daggerdbl
/circumflex/perthousand/Scaron/guilsinglleft
/OE/Zcaron/asciicircum/minus/lslash/quoteleft
/quoteright/quotedblleft/quotedblright/bullet
/endash/emdash/tilde/trademark/scaron
/guilsinglright/oe/zcaron/asciitilde
/Ydieresis/space/exclamdown/cent/sterling
/currency/yen/brokenbar/section/dieresis
/copyright/ordfeminine/guillemotleft
/logicalnot/hyphen/registered/macron/degree
/plusminus/twosuperior/threesuperior/acute
/mu/paragraph/periodcentered/cedilla
/onesuperior/ordmasculine/guillemotright
/onequarter/onehalf/threequarters
/questiondown/Agrave/Aacute/Acircumflex
/Atilde/Adieresis/Aring/AE/Ccedilla/Egrave
/Eacute/Ecircumflex/Edieresis/Igrave/Iacute
/Icircumflex/Idieresis/Eth/Ntilde/Ograve
/Oacute/Ocircumflex/Otilde/Odieresis/multiply
/Oslash/Ugrave/Uacute/Ucircumflex/Udieresis
/Yacute/Thorn/germandbls/agrave/aacute
/acircumflex/atilde/adieresis/aring/ae
/ccedilla/egrave/eacute/ecircumflex/edieresis
/igrave/iacute/icircumflex/idieresis/eth
/ntilde/ograve/oacute/ocircumflex/otilde
/odieresis/divide/oslash/ugrave/uacute
/ucircumflex/udieresis/yacute/thorn
/ydieresis]
>> endobj
5 0 obj <<
/Type /Font
/Subtype /Type1
/Encoding 7 0 R
/BaseFont /Times-Roman
>> endobj
6 0 obj <<
/Type /Pages
/Count 1
/Kids [2 0 R]
>> endobj
8 0 obj <<
/Type /Catalog
/Pages 6 0 R
>> endobj
9 0 obj <<
/Creator (TeX)
/Producer (pdfTeX-0.12r)
/CreationDate (D:19981205172300)
>> endobj
xref
0 10
0000000000 65535 f
0000000242 00000 n
0000000308 00000 n
0000000009 00000 n
0000000223 00000 n
0000002238 00000 n
0000002326 00000 n
0000000420 00000 n
0000002383 00000 n
0000002432 00000 n
trailer
<<
/Size 10
/Root 8 0 R
/Info 9 0 R
>>
startxref
2526
%%EOF
```

Figure 1 The title page of the stand-alone version of this manual represents the plain TEX coded text "Welcome to pdfTEX!"

catalog. In this catalog the viewer can find the page list (in our example we have only one page).

The trailer also holds an `/Info` entry, which tells a bit more about the document. Just follow the thread:

`/Root` ⟶ object 8 ⟶ `/Pages` ⟶ object 6 ⟶ `/Kids` ⟶ object 2 ⟶ page content

As soon as we add annotations, a fancy word for hyperlinks and alike, some more entries are present in the catalog. We invite users to take a look at the PDF code of this file to get an impression of that.

The page content is a stream of drawing operations. Such a stream can be compressed, where the level of compression can be set with `\pdfcompresslevel`. Let's take a closer look at this stream. First there is a transformation matrix: six numbers followed by `cm`. As in POSTSCRIPT, the operator comes after the operands. Between `BT` and `ET` comes the text. A font switch can be recognized as `/F...` The actual text goes between `()` so that it creates a POSTSCRIPT string. When one analyzes a file produced by a less sophisticated typesetting engine, whole sequences of words can be recognized. In TEX however, the text comes out rather fragmented, mainly because a lot of kerning takes place. Because viewers can search in these streams, one can imagine that the average TEX produced files becomes more difficult as soon as the typesetting engine does a better job; TEX cannot do less.

This one page example uses an Adobe Times Roman font. This is one of the 14 fonts that is always present in the viewer application, and is called a base font. However, when we use for instance Computer Modern Roman, we have to make sure that this font is available, and the best way to do this is to embed it in the file. Just let your eyes follow the object thread and see how a font is described. The only thing missing in this example is the (partially) embedded glyph description file, which for the base fonts is not needed.

In this simple file, we don't specify in what way the file should be opened, for instance full screen or clipped. A closer look at the page object (`/Type /Page`) shows that a mediabox is part of the page description. A mediabox acts like the bounding box in a POSTSCRIPT file. PDFTEX users have access to this object by `\pdfpageattr`.

Although in most cases macro packages will shield users from these internals, PDFTEX provides access to many of the entries described here, either automatically by translating the TEX data structures into PDF ones, or manully by pushing entries to the catalog, page, info or self created objects. Those who, after this introduction, feel uncomfortable in how to proceed, are advised to read on but skip section 6. Before we come to that section, we will describe how to get started with PDFTEX.

3 Getting started

This section describes the steps needed to get PDFTEX running on a system where PDFTEX is not yet installed. Some TEX distributions have PDFTEX as a component, like TETEX, FPTEX, MIKTEX and CMACTEX, so when you use one of them, you don't need to bother with the PDFTEX installation. Note that the installation description in this manual is WEB2C–specific.

For some years there has been a 'moderate' successor to TEX available, called E-TEX. Because the main stream macro packages start supporting this welcome extension, PDFTEX is also available as PDF-E-TEX. Although in this document we will speak of PDFTEX, we advise users to use PDF-E-TEX when available. That way they get the best of all worlds and are ready for the future.

3.1 Getting sources and binaries

The latest sources of PDFTEX are distributed together with precompiled binaries of PDFTEX for some platforms, including Linux[2], SGI IRIX, Sun SPARC Solaris and MS-DOS (DJGPP).[3] The primary location where one can fetch the source code is:

`ftp://ftp.cstug.cz/pub/tex/local/cstug/thanh/pdftex-testing/latest`

2 The Linux binary is compiled for the new `libc-6` (GNU `glibc-2.0`), which will not run for users of older Linux installations still based on `libc-5`.

3 The DJGPP version is built by the DJGPP 2.0 cross–compiler on Linux.

For WIN32 systems (Windows 95, Windows NT) there are two packages that contain PDFTEX, both in ctan:systems/win32: FPTEX, maintained by Fabrice Popineau, popineau@ese-metz.fr, and MIKTEX by Christian Schenk, cschenk@berlin.snafu.de.

A binary version of PDFTEX for the AMIGA comes with the AMIWEB2C distribution (ctan:systems/amiga/amiweb2c) by Andreas Scherer (andreas.scherer@pobox.com). For the MACINTOSH there is CMACTEX.

3.2 Compiling

If there is no precompiled binary of PDFTEX for your system, you need to build PDFTEX from sources. The compilation is expected to be easy on UNIX–like systems and can be described best by example. Assuming that all needed files are downloaded to $HOME/pdftex, on a UNIX system the following steps are needed to compile PDFTEX:

```
cd \$HOME/pdftex
gunzip < web-7.2.tar.gz | tar xvf -
gunzip < web2c-7.2.tar.gz | tar xvf -
gunzip < pdftex.tar.gz | tar xvf -
mv pdftexdir web2c-7.2/web2c
cd ./web2c-7.2
./configure
cd ./web2c
make pdftex
```

If you happen to have a previously configured source tree and just install a new version of PDFTEX, you can avoid running configure from the top–level directory. It's quicker to run config.status, which will just regenerate the Makefile's based on config.cache:

```
cd web2c-7.2/web2c
sh config.status
make pdftex
```

Apart from the binary of PDFTEX the compilation also produces several other files which are needed for running PDFTEX:

pdftex.pool The pool file, needed for creating formats, located in web2c-7.2/web2c

texmf.cnf WEB2C run–time configuration file, located in web2c-7.2/kpathsea

ttf2afm An external program to generate AFM files from TrueType fonts, located in web2c-7.2/web2c/pdftexdir

Precompiled binaries are included in the ZIP archive pdftex.zip.

3.3 Getting PDFTEX–specific platform–independent files

Apart from above–mentioned files, there is another ZIP archive (pdftexlib-0.12.zip) in the PDFTEX distribution which contains platform–independant files required for running PDFTEX:

- configuration file: pdftex.cfg
- encoding vectors: *.enc
- map files: *.map
- macros: *.tex

Unpacking this archive —don't forget the -d option when using pkunzip— will create a texmf tree containing PDFTEX–specific files.

3.4 Placing files

The next step is to place the binaries somewhere in PATH. If you want to use LaTeX, you also need to make a copy (or symbolic link) of pdftex and name it pdflatex. The files texmf.cnf and pdftex.pool and the directory texmf, created by unpacking the file pdftexlib-0.12.zip, should be moved to the 'appropriate' place (see below).

3.5 Setting search paths

WEB2C–based programs, including PDFTEX, use the WEB2C run–time configuration file called texmf.cnf. This file can be found via the user–set environment variable TEXMFCNF or via the compile–time default value if the former is not set. It is strongly recommended to use the first option. Next you need to edit texmf.cnf so PDFTEX can find all necessary files. Usually one has to edit TEXMFS and maybe some of the next variables. When running PDFTEX, some extra search paths are used beyond those normally requested by TEX itself:

VFFONTS	the path where PDFTEX looks for virtual fonts
T1FONTS	the path where PDFTEX looks for Type1 fonts
TTFONTS	the path where PDFTEX looks for TrueType fonts
PKFONTS	the path where PDFTEX looks for PK fonts
TEXPSHEADERS	the path where PDFTEX looks for the configuration file pdftex.cfg, font mapping files (*.map), encoding files (*.enc), and pictures

3.6 The PDFTEX configuration file

One has to keep in mind that, opposed to DVI output, there is no postprocessing stage. This has several rather fundamental consequences, like one–pass graphic and font inclusion. When TEX builds a page, the macro package used quite certain has a concept of page dimensions, which is not the same as paper dimensions. The reference point of the page is the top–left corner.

Most DVI postprocessors enable the user to specify the paper size, which often defaults to 'A4' or 'letter'. In most cases it does not harm that much to mix the two, because one will seldom put too small paper in the printer. And, if one does, one will certainly not do that a second time. In PDF the paper size is part of the definition. This means that everything that is off page, is clipped off, it simply disappears. Even worse, just like in a POSTSCRIPT file, the reference point is in the lower corner, which is opposite to DVI's reference point.

And so, we've found one of the main reasons why PDFTEX explicitly needs to know the paper dimensions. These dimensions can either be passed using the so called configuration file, or by using the primitives provided for this purpose. In this respect, the PDFTEX configuration file can be compared to configuration files that come with DVI postprocessors and/or command line options. Both contain information on the paper used, the fonts to be included and optimizations to be applied.

When PDFTEX starts, it reads the WEB2C configuration file as well as the PDFTEX configuration file called pdftex.cfg, searched for in the TEXPSHEADERS path. As WEB2C systems commonly specify a 'private' tree for PDFTEX where configuration and map files are located, this allows individual users or projects to maintain customized versions of the configuration file.

The configuration file sets default values for the following parameters, all of which can be over–ridden in the TEX source file:

output_format This integer parameter specifies whether the output format should be DVI or PDF. A positive value means PDF output, otherwise we get DVI output.

compress_level This integer parameter specifies the level of text and in–line graphics compression. PDFTEX uses ZIP compression as provided by `zlib`. A value of 0 means no compression, 1 means fastest, 9 means best, 2..8 means something in between. Just set this value to 9, unless there is a good reason to do otherwise — 0 is great for testing macros that use `\pdfliteral`.

decimal_digits This integer specifies the preciseness of real numbers in PDF page descriptions. It gives the maximal number of decimal digits after the decimal point of real numbers. Valid values are in range 0..5. A higher value means more precise output, but also results in a much larger file size and more time to display or print. In most cases the optimal value is 2. This parameter does not influence the precision of numbers used in raw PDF code, like that used in `\pdfliterals` and annotation action specifications.

image_resolution When PDFTEX is not able to determine the natural dimensions of an image, it assumes a resolution of type 72 dots per inch. Use this variable to change this default value.

page_width & page_height These two dimension parameters specify the output medium dimensions (the paper, screen or whatever the page is put on). If they are not specified, the page width is calculated as $w_{\text{box being shipped out}} + 2 \times (\text{horigin} + \texttt{\textbackslash hoffset})$. The page height is calculated in a similar way.

horigin & vorigin These dimension parameters can be used to set the offset of the TEX output box from the top left corner of the 'paper'.

map This entry specifies the font mapping file, which is similar to those used by many DVI to POSTSCRIPT drivers. More than one map file can be specified, using multiple `map` lines. If the name of the map file is prefixed with a `+`, its values are appended to the existing set, otherwise they replace it. If no map files are given, the default value `psfonts.map` is used.

A typical `pdftex.cfg` file looks like this, setting up output for A4 paper size and the standard TEX offset of 1 inch, and loading two map files for fonts:

```
output_format    1
compress_level   0
decimal_digits   2
image_resolution 300
page_width       210mm
page_height      297mm
horigin          1in
vorigin          1in
map              standard.map
map              +cm.map
```

Dimensions can be specified as `true`, which makes them immune for magnification (when set). The previous example settings, apart from `map`, can also be set during a TEX run. This leaves a special case:

include_form_resources Sometimes embedded PDF illustrations can pose viewers for problems. When set to 1, this variable makes PDFTEX take some precautions. Forget

about it when you never encounteres problems. When all the programs you use conform to the PDF specifications, you will never need to set this variable.

3.7 Creating formats

Formats for PDFTEX are created in the same way as for TEX. For plain TEX and LATEX it looks like:

```
pdftex -ini -fmt=pdftex   plain      \dump
pdftex -ini -fmt=pdflatex latex.ltx
```

In CONTEXT the generation depends on the interface used. A format using the english user interface is generated with

```
pdftex -ini -fmt=cont-en cont-en
```

When properly set up, one can also use the CONTEXT command line interface TEXEXEC to generate one or more formats, like:

```
texexec --make en
```

for an english format, or

```
texexec --make --tex=pdfetex en de
```

for an english and german one, using PDF-E-TEX. Indeed, if there is PDFTEX as well as PDF-E-TEX, use it! Whatever macro package used, the formats should be placed in the TEXFORMATS path. We strongly recommend to use PDF-E-TEX, if only because the main stream macro packages (will) use it.

3.8 Testing the installation

When everything is set up, you can test the installation. In the distribution there is a plain TEX test file example.tex. Process this file by saying:

```
pdftex example
```

If the installation is ok, this run should produce a file called example.pdf. The file example.tex is also a good place to look for how to use PDFTEX's new primitives.

3.9 Common problems

The most common problem with installations is that PDFTEX complains that something cannot be found. In such cases make sure that TEXMFCNF is set correctly, so PDFTEX can find texmf.cnf. The next best place to look/edit is the file texmf.cnf. When still in deep trouble, set KPATHSEA_DEBUG=255 before running PDFTEX or run PDFTEX with option -k 255. This will cause PDFTEX to write a lot of debugging information that can be useful to trace problems. More options can be found in the WEB2C documentation.

Variables in texmf.cnf can be overwritten by environment variables. Here are some of the most common problems you can encounter when getting started:

- I can't read tex.pool; bad path?
  TEXMFCNF is not set correctly and so PDFTEX cannot find texmf.cnf, or TEXPOOL in texmf.cnf doesn't contain a path to the pool file pdftex.pool.
- You have to increase POOLSIZE.
  PDFTEX cannot find texmf.cnf, or the value of pool_size specified in texmf.cnf is not large enough and must be increased. If pool_size is not specified in texmf.cnf then you can add something like

  ```
  pool_size = 500000
  ```

- `I can't find the format file 'pdftex.fmt'!`
  `I can't find the format file 'pdflatex.fmt'!`
  Format is not created (see above how to do that) or is not properly placed. Make sure that `TEXFORMATS` in `texmf.cnf` contains the path to `pdftex.fmt` or `pdflatex.fmt`.
- `Fatal format file error; I'm stymied.`
  This appears if you forgot to regenerate the `.fmt` files after installing a new version of the PDFTeX binary and `pdftex.pool`.
- `TEX.POOL doesn't match; TANGLE me again!`
  `TEX.POOL doesn't match; TANGLE me again (or fix the path).`
  This might appear if you forgot to install the proper `pdftex.pool` when installing a new version of the PDFTeX binary.
- PDFTeX cannot find the configuration file `pdftex.cfg`, one or more map files (`*.map`), encoding vectors (`*.enc`), virtual fonts, Type 1 fonts, TrueType fonts or some image file.

  Make sure that the required file exists and the corresponding variable in `texmf.cnf` contains a path to the file. See above which variables PDFTeX needs apart from the ones TeX uses.

Normally the page content takes one object. This means that one seldom finds more than a few hundred objects in a simple file. This document for instance uses about 300 objects. In demanding applications this number can grow quite rapidly, especially when one uses a lot of widget annotations, shared annotations or other shared things. In these situations one can enlarge PDFTeX's internal object table by adding a line in `texmf.cfg`, for instance:

```
obj_tab_size = 400000
```

4 Macro packages supporting PDFTeX

When producing DVI output, for which one can use PDFTeX as well as any other TeX, part of the job is delegated to the DVI postprocessor, either by directly providing this program with commands, or by means of `\specials`. Because PDFTeX directly produces the final format, it has to everything itself, from handling color, graphics, hyperlink support, font–inclusion, upto page imposition and page manipulation.

As a direct result, when one uses a high level macro package, the macros that take care of these features have to be set up properly. Specials for instance make no sense at all. Actually being a comment understood by DVI postprocessors —given that the macro package speaks the specific language of this postprocessor— a `\special` would end up as just a comment in the PDF file, which is of no use. Therefore, `\special` issues a warning when PDFTeX is in PDF mode.

When one wants to get some insight to what extend PDFTeX specific support is needed, one can start a file by saying:

```
\pdfoutput=1 \let\special\message
```

or, if this leads to confusion,

```
\pdfoutput=1 \def\special#1{\write16{special: #1}}
```

And see what happens. As soon as one 'special' message turns up, one knows for sure that some kind of PDFTeX specific support is needed, and often the message itself gives a indication of what is needed.

Currently all main stream macro packages offer PDFTeX support in one way or the other. When using such a package, it makes sense to turn on this support in the appropriate way, otherwise one cannot be sure if things are set up right. Remember that for

instance the page and paper dimensions have to be taken care of, and only the macro package knows the details.

- For LaTeX users, Sebastian Rahtz' `hyperref` package has substantial support for pdfTeX, and provides access to most of its features. In the simplest case, the user merely needs to load `hyperref` with a `pdftex` option, and all cross–references will be converted to PDF hypertext links. PDF output is automatically selected, compression is turned on, and the page size is set up correctly. Bookmarks are created to match the table of contents.
- The standard LaTeX `graphics` and `color` packages have `pdftex` options, which allow use of normal color, text rotation, and graphics inclusion commands.
- The ConTeXt macro package by Hans Hagen (pragma@wxs.nl) has very full support for pdfTeX in its generalized hypertext features. Support for pdfTeX is implemented as a special driver, and is invoked by saying `\setupoutput[pdftex]` or feeding TeXexec with the `--pdf` option.
- Hypertexted PDF from `texinfo` documents can be created with `pdftexinfo.tex`, which is a slight modification of the standard `texinfo` macros. This file is part of the pdfTeX distribution.
- A similar modification of `webmac.tex`, called `pdfwebmac.tex`, allows production of hypertext'd PDF versions of programs written in WEB. This is also part of the pdfTeX distribution.

Some nice samples of pdfTeX output can be found on the TUG web server, at `http://www.tug.org/applications/pdftex` and `http://www.ntg.nl/context`.

5 Setting up fonts

pdfTeX can work with Type 1 and TrueType fonts, but a source must be available for all fonts used in the document, except for the 14 base fonts supplied by Acrobat Reader (Times, Helvetica, Courier, Symbol and Dingbats). It is possible to use MetaFont–generated fonts in pdfTeX— but it is strongly recommended not to use MetaFont–fonts if an equivalent is available in Type 1 or TrueType format, if only because bitmap Type 3 fonts render very poorly in Acrobat Reader. Given the free availability of Type 1 versions of all the Computer Modern fonts, and the ability to use standard PostScript fonts, most TeX users should be able to experiment with pdfTeX.

5.1 Map files

pdfTeX reads the map files, specified in the configuration file, see section 3.6, in which reencoding and partial downloading for each font are specified. Every font needed must be listed, each on a separate line, except PK fonts. The syntax of each line is similar to `dvips` map files[4] and can contain up to the following (some are optional) fields: texname, basename, fontflags, fontfile, encodingfile and special. The only mandatory is texname and must be the first field. The rest is optional, but if basename is given, it must be the second field. Similarly if fontflags is given it must be the third field (if basename is present) or the second field (if basename is left out). It is possible to mix the positions of fontfile, encodingfile and special, however the first three fields must be given in fixed order.

texname sets the name of the TFM file. This name must be given for each font.

basename sets the base (PostScript) font name. If not given then it will be taken

4 `dvips` map files can be used with pdfTeX without problems.

from the font file. Specifying a name that doesn't match the name in the font file will cause PDFTEX to write a warning, so it is best not to have this field specified if the font resource is available, which is the most common case. This option is primarily intended for use of base fonts and for compatibility with `dvips` map files.

fontflags specify some characteristics of the font. The next description of these flags are taken, with a slight modification, from the PDF Reference Manual (the section on Font Descriptor Flags).

The value of the flags key in a font descriptor is a 32–bit integer that contains a collection of boolean attributes. These attributes are true if the corresponding bit is set to 1. Table 1 specifies the meanings of the bits, with bit 1 being the least significant. Reserved bits must be set to zero.

bit position	semantics
1	Fixed-width font
2	Serif font
3	Symbolic font
4	Script font
5	Reserved
6	Uses the Adobe Standard Roman Character Set
7	Italic
8–16	Reserved
17	All-cap font
18	Small-cap font
19	Force bold at small text sizes
20–32	Reserved

Table 1 The meaning of flags in the font descriptor.

All characters in a *fixed–width* font have the same width, while characters in a proportional font have different widths. Characters in a *serif font* have short strokes drawn at an angle on the top and bottom of character stems, while sans serif fonts do not have such strokes. A *symbolic font* contains symbols rather than letters and numbers. Characters in a *script font* resemble cursive handwriting. An *all–cap* font, which is typically used for display purposes such as titles or headlines, contains no lowercase letters. It differs from a *small–cap* font in that characters in the latter, while also capital letters, have been sized and their proportions adjusted so that they have the same size and stroke weight as lowercase characters in the same typeface family.
Bit 6 in the flags field indicates that the font's character set conforms the Adobe Standard Roman Character Set, or a subset of that, and that it uses the standard names for those characters.
Finally, bit 19 is used to determine whether or not bold characters are drawn with extra pixels even at very small text sizes. Typically, when characters are drawn at small sizes on very low resolution devices such as display screens, features of bold characters may appear only one pixel wide. Because this is the minimum feature width on a pixel–based device, ordinary non–bold characters also appear with one–pixel wide features, and cannot be distinguished from bold characters. If bit 19 is set, features of bold characters may be thickened at small text sizes.

If the font flags are not given, PDFTEX treats it as being 4, a symbolic font. If you do not

know the correct value, it would be best not to specify it, as specifying a bad value of font flags may cause troubles in viewers. On the other hand this option is not absolutely useless because it provides backward compatibility with older map files (see the fontfile description below).

fontfile sets the name of the font source file. This must be a Type 1 or TrueType font file. The font file name can be preceded by one or two special characters, which says how the font file should be handled.

- If it is preceded by a < the font file will be partly downloaded, which means that only used glyphs (characters) are embedded to the font. This is the most common use and is strongly recommended for any font, as it ensures the portability and reduces the size of the PDF output. Partial fonts are included in such a way that name and cache clashes are minimalized.
- In case the font file name is preceded by a double <<, the font file will be included entirely — all glyphs of the font are embedded, including the ones that are not used in the document. Apart from causing large size PDF output, this option may cause troubles with TrueType fonts too, so it is not recommended. It might be useful in case the font is untypical and can not be subsetted well by PDFTEX. Beware: some font vendors forbid full font inclusion.
- In case nothing preceded the font file name, the font file is read but nothing is embedded, only the font parameters are extracted to generate the so–called font descriptor, which is used by Acrobat Reader to simulate the font if needed. This option is useful only when you do not want to embed the font (i.e. to reduce the output size), but wish to use the font metrics and let Acrobat Reader generate instances that look close to the used font in case the font resource is not installed on the system where the PDF output will be viewed or printed. To use this feature the font flags must be specified, and it must have the bit 6 set on, which means that only fonts with the Adobe Standard Roman Character Set can be simulated. The only exception is in case of Symbolic font, which is not very useful.
- If the font file name is preceded by a !, the font is not read at all, and is assumed to be available on the system. This option can be used to create PDF files which do not contain embedded fonts. The PDF output then works only on systems where the resource of the used font is available. It's not very useful for document exchange, as the PDF is not 'portable' at all. On the other hand it is very useful when you wish to speed up running of PDFTEX during interactive work, and only in a final version embed all used fonts. Don't over–estimate gain in speed and when distributing files, always embed the fonts! This feature requires Acrobat Reader to have access to installed fonts on the system. This has been tested on Win95 and UNIX (Solaris).

Note that the standard 14 fonts are never downloaded, even when they are specified to be downloaded in map files.

encoding specifies the name of the file containing the external encoding vector to be used for the font. The file name may be preceded by a <, but the effect is the same. The format of the encoding vector is identical to that used by `dvips`. If no encoding is specified, the font's built–in default encoding is used. It may be omitted if you are sure that the font resource has the correct built–in encoding. In general this option is highly preferred and is required when subsetting a TrueType font.

special instructions can be used to manipulate fonts similar to the way `dvips` does. Currently only the keyword `SlantFont` is interpreted, other instructions are just ignored.

If a used font is not present in the map files, first PDFTeX will look for a source with suffix `.pgc`, which is a so–called PGC source (PDF Glyph Container)[5]. If no PGC source is available, PDFTeX will try to use PK fonts in a normal way as DVI drivers do, on–the–fly creating PK fonts if needed.

Lines containing nothing apart from texname stand for scalable Type 3 fonts. For scalable fonts as Type 1, TrueType and scalable Type 3 font, all the fonts loaded from a TFM at various sizes will be included only once in the PDF output. Thus if a font, let's say `csr10`, is described in one of the map files, then it will be treated as scalable. As a result the font source for csr10 will be included only once for `csr10`, `csr10 at 12pt` etc. So PDFTeX tries to do its best to avoid multiple downloading of identical font sources. Thus vector PGC fonts should be specified as scalable Type 3 in map files like:

```
csr10
```

It doesn't hurt much if a scalable Type 3 font is not given in map files, except that the font source will be downloaded multiple times for various sizes, which causes a much larger PDF output. On the other hand if a font is in the map files is defined as scalable Type 3 font and its PGC source is not scalable or not available, PDFTeX will use PK font instead; the PDF output is still valid but some fonts may look ugly because of the scaled bitmap.

To summarize this rather confusing story, we include some sample lines.

Use a built–in font with font–specific encoding, i.e. neither a download font nor an external encoding is given. A SlantFont is specified similarly as for `dvips`.

```
psyr   Symbol
psyro  Symbol        ".167 SlantFont"
pzdr   ZapfDingbats
```

Use a built–in font with an external encoding. The < preceded encoding file may be left out.

```
ptmr8r   Times-Roman   <8r.enc
ptmri8r  Times-Italic  <8r.enc
ptmro8r  Times-Roman   <8r.enc  ".167 SlantFont"
```

Use a partially downloaded font with an external encoding:

```
putr8r   Utopia-Regular  <8r.enc <putr8a.pfb
putri8r  Utopia-Italic   <8r.enc <putri8a.pfb
putro8r  Utopia-Regular  <8r.enc <putr8a.pfb   ".167 SlantFont"
```

Use some faked font map entries:

```
logo8      <logo8.pfb
logo9      <logo9.pfb
logo10     <logo10.pfb
logosl8    <logo8.pfb     ".25 SlantFont"
logosl9    <logo9.pfb     ".25 SlantFont"
logosl10   <logosl10.pfb
logobf10   <logobf10.pfb
```

Use an ASCII subset of OT1 and T1:

5 This is a text file containing a PDF Type 3 font, created by METAPOST using some utilities by Hans Hagen. In general PGC files can contain whatever allowed in PDF page description, which may be used to support fonts that are not available in METAFONT. At the moment PGC fonts are not very useful, as vector Type 3 fonts are not displayed very well in Acrobat Reader, but it may be more useful when Type 3 font handling gets better.

```
ectt1000  cmtt10  <cmtt10.map  <tex256.enc
```

Download a font entirely without reencoding:

```
pgsr8r GillSans  <<pgsr8a.pfb
```

Partially download a font without reencoding:

```
pgsr8r GillSans  <pgsr8a.pfb
```

Do not read the font at all — the font is supposed to be installed on the system:

```
pgsr8r GillSans  !pgsr8a.pfb
```

Entirely download a font with reencoding:

```
pgsr8r GillSans  <<pgsr8a.pfb  8r.enc
```

Partially download a font with reencoding:

```
pgsr8r GillSans  <pgsr8a.pfb  8r.enc
```

Sometimes we do not want to include a font, but need to extract parameters from the font file and reencode the font as well. This only works for fonts with Adobe Standard Encoding. The font flags specify how such a font looks like, so Acrobat Reader can generate similar instance if the font resource is not available on the target system.

```
pgsr8r GillSans  32  pgsr8a.pfb  8r.enc
```

A TrueType font can be used in the same way as a Type 1 font:

```
verdana8r Verdana  <verdana.ttf  8r.enc
```

5.2 TrueType fonts

As mentioned above, PDFTEX can work with TrueType fonts. Defining TrueType files is similar to Type 1 font. The only extra thing to do with TrueType is to create a TFM file. There is a program called `ttf2afm` in the PDFTEX distribution which can be used to extract AFM from TrueType fonts. Usage is simple:

```
ttf2afm -e <encoding vector> -o <afm outputfile> <ttf input file>
```

A TrueType file can be recognized by its suffix `ttf`. The optional encoding specifies the encoding, which is the same as the encoding vector used in map files for PDFTEX and `dvips`. If the encoding is not given, all the glyphs of the AFM output will be mapped to `/.notdef`. `ttf2afm` writes the output AFM to standard output. If we need to know which glyphs are available in the font, we can run `ttf2afm` without encoding to get all glyph names. The resulting AFM file can be used to generate a TFM one by applying `afm2tfm`.

To use a new TrueType font the minimal steps may look like below. We suppose that `test.map` is included in `pdftex.cfg`.

```
ttf2afm -e 8r.enc -o times.afm times.ttf
afm2tfm times.afm -T 8r.enc
echo "times TimesNewRomanPSMT <times.ttf <8r.enc" >>test.map
```

The POSTSCRIPT font name (`TimesNewRomanPSMT`) is reported by `afm2tfm`, but from PDFTEX version 0.12l onwards it may be left out.

The SlantFont transformation also works for TrueType fonts.

6 New primitives

Here follows a short description of new primitives added by PDFTEX. One way to learn more about how to use these primitives is to have a look at the file `example.tex` in the

PDFTeX distribution. Each PDFTeX specific primitive is prefixed by \pdf.

6.1 Document setup

6.1.1 \pdfoutput = *number*

This Integer parameter specifies whether the output format should be DVI or PDF. Positive value means PDF output, otherwise DVI output. This parameter cannot be specified after shipping out the first page. In other words, this parameter must be set before PDFTeX ships out the first page if we want PDF output. This is the only one parameter that must be set to produce PDF output. All others are optional.

When PDFTeX starts complaining about specials, one can be sure that the macro package is not aware of this mode. A simple way of making macros PDFTeX aware is:

```
\ifx\pdfoutput\undefined \newcount\pdfoutput \fi

\ifcase\pdfoutput DVI CODE \else PDF CODE \fi
```

However, there are better ways to handle these things.

6.1.2 \pdfcompresslevel = *number*

This integer parameter specifies the level of text compression via zlib. Zero means no compression, 1 means fastest, 9 means best, 2..8 means something in between. A value out of this range will be adjusted to the nearest meaningful value. Use a value of 9 for normal runs.

6.1.3 \pdfpagewidth = *dimension*

This dimension parameter specifies the page width of the PDF output. If not given then the page width will be calculated as mentioned above. Like the next one, this value replaces the value set in the configuration file. When part of the page falls of the paper or screen, you can be rather sure that this parameter is set wrong.

6.1.4 \pdfpageheight = *dimension*

Similar to the previous one, this dimension parameter specifying the page height of the PDF output. If not given then the page height will be calculated as mentioned above.

6.1.5 \pdfpagesattr = *tokens*

Use this token list parameter to specify optional attributes common for all pages of the PDF output file. Some examples of attributes are /MediaBox, the rectangle specifying the natural size of the page, /CropBox, the rectangle specifying the region of the page being displayed and printed, and /Rotate, the number of degrees (in multiples of 90) the page should be rotated clockwise when it is displayed or printed.

6.1.6 \pdfpageattr = *tokens*

This is similar to \pdfpagesattr, but it takes priority to the former one. It can be used to overwrite any attribute given by \pdfpagesattr for individual pages.

6.2 The document info and catalog

6.2.1 \pdfinfo {*info keys*}

This allows the user to add information to the document info section; if this information is provided, it can be extracted by Acrobat Reader (version 3.1: menu option Document Information, General). The {*info keys*} is a set of data pairs, a key and a value. The key names are preceded by a /, and the values, being strings, are given between parentheses. All keys are optional. Possible keys are /Author, /CreationDate (defaults to current

date), /ModDate, /Creator (defaults to TeX), /Producer (defaults to pdfTeX), /Title, /Subject, and /Keywords.

/CreationDate and /ModDate are expressed in the form D:YYYYMMDDhhmmss, where YYYY is the year, MM is the month, DD is the day, hh is the hour, mm is the minutes, and ss is the seconds.

Multiple appearances of \pdfinfo will be concatenated to only one. If a key is given more than once, then the first appearance will take priority. An example of use of \pdfinfo may look like:

```
\pdfinfo
  { /Title        (example.pdf)
    /Creator      (TeX)
    /Producer     (pdfTeX 0.15a)
    /Author       (Tom and Jerry)
    /CreationDate (D:19980212201000)
    /ModDate      (D:19980212201000)
    /Subject      (Example)
    /Keywords     (pdfTeX) }
```

6.2.2 \pdfcatalog { *catalog keys* } ⟨openaction *action*⟩

Similar to the document info section is the document catalog, where keys are /URI, which provides the base URL of the document, and /PageMode determines how Acrobat displays the document on startup. The possibilities for the latter are explained in Table 2:

value	meaning
/UseNone	neither outline nor thumbnails visible
/UseOutlines	outline visible
/UseThumbs	thumbnails visible
/FullScreen	full–screen mode

Table 2 Supported /PageMode values.

In full–screen mode, there is no menu bar, window controls, nor any other window present. The default setting is /UseNone.

The openaction is the action provided when opening the document and is specified in the same way as internal links, see section 6.7. Instead of using this method, one can also write the open action directly into the catalog.

6.2.3 \pdfnames { *text* }

Inserts the text to /Names array. The text must be conform to the specifications as laid down in the PDF Reference Manual, otherwise the document can be invalid.

6.3 Fonts

6.3.1 \font ... *number* stretch *number* shrink *number* step *number*

Although still in an experimental stage, and therefore subjected to changes, the next extension to the TEX primitive font is worth mentioning.

```
\font\somefont=somefile at 10pt stretch 30 shrink 20 step 10
```

The stretch 30 shrink 20 step 5 means as much as: “hey TEX, when things are going to bad, you may stretch the glyphs in this font as much as 3% or shrink them by 2%”. Because PDFTEX uses internal datastructures with fixed widths, each additional width also means an additional font. For practical reasons PDFTEX uses discrete steps, in this

example a 1% one. This means that for font `somefile` upto 6 different alternatives are used. When no `step` is specified, 0.5% steps are used.

Roughly spoken, the trick is as follows. Consider a text typeset in triple column mode. When TeX cannot break a line in the appropriate way, the unbreakable parts of the word will stick into the margin. When pdfTeX notes this, it will try to scale (shrink) the glyphs in that line using fixed steps, until the line fits. When lines are too spacy, the opposite happens: pdfTeX starts scaling (stretching) the glyphs until the whitespace gaps are acceptable.

The additional fonts are named as `somefile+10` or `somefile-15`, and TFM files with these names and appropriate dimensions must be available. So, each scaled font must have its own TFM file! When no TFM file can be found, pdfTeX will try to generate it by executing the script `mktextfm`, where available and when supported.

The mechanism is inspired on an optimization introduced first by Herman Zapf, which in itself goes back to optimizations used in the early days of typesetting: use different glyphs to optimize the greyness of a page. So, there are many, slightly different a's, e's, etc. For practical reasons pdfTeX does not use such huge glyph collections; it uses horizontal scaling instead. This is sub–optimal, and for many fonts, sort of offending to the design. But, when using PDF, it's not that illogical at all: PDF viewers use so called Multiple Master fonts when no fonts are embedded and/or can be found on the target system. Such fonts are designed to adapt their design to the different scaling parameters. It is up to the user to determine to what extend mixing slightly remastered fonts can be used without violating the design. Think of an O: when simply stretched, the vertical part of the glyph becomes thicker, and looks incompatible to an unscaled original. In a multiple master, one can decide to stretch but keep this thickness compatible.

6.3.2 `\pdfadjustspacing` *number*

The output that pdfTeX produces is pretty compatible with the normal TeX output: TeX's typesetting engine is normally unchanged, because the optimization described here is turned off by default.

At this moment there are two methods provided. When `\pdfadjustspacing` is set to 1, stretching is applied after after TeX's normal paragraph breaking routines have broken the paragraph into lines. In this case, line breaks are indentical to standard TeX behaviour.

When set to 2, the width changes that are the result of stretching and shrinking are taken into account while the paragraph is broken into lines. In this case, line breaks are likely to be different from those of standard TeX. In fact, paragraphs may even become longer or shorter!

Both alternatives use the extended collection of TFM files that are related to the `stretch` and `shrink` settings as described in the previous section.

6.3.3 `\efcode` *number*

We didn't yet tell the whole story. One can imagine that some glyphs are more sensitive to scaling than others. The `\efcode` primitive can be used to influence the stretchability of a glyph. The syntax is similar to `\sfcode`, and defaults to 1000, meaning 100%.

```
\efcode`A=2500
\efcode`O=0
```

In this example an A may stretch 2.5 times as much as normal and the O is not to be stretched at all. The minimum and maximum stretch is however bound by the font specification, otherwise one would end up with more fonts inclusions than comfortable.

6.4 Graphics inclusion

6.4.1 `\pdfimage width` *dimension* `height` *dimension* `depth` *dimension* {*filename*}
Inserts an image, optionally changing width, height, depth or any combination of them. Default values are zero for depth and 'running' for height and width. If all of them are given, the image will be scaled to fit the specified values. If some of them (but not all) are given, the rest will be set to a value corresponding to the remaining ones so as to make the image size to yield the same proportion of $width : (height + depth)$ as the original image size, where depth is treated as zero. If none of them is given then the image will take its natural size. An image inserted at its natural size often has a resolution of 72 dots per inch in the output file, but some images may contain data specifying the image resolution, and in such a case the image will be scaled to the correct resolution.

The filename of the image must appear after the optional dimension parameters. The dimension of the image can be accessed by enclosing the `\pdfimage` command to a box and checking the dimensions of the box:

```
\setbox0=\hbox{\pdfimage {somefile.png}}
```

Now we can use `\wd0` and `\ht0` to question the natural size of the image as determined by PDFTEX. When dimensions are specified before the `{somefile.pdf}`, the graphic is scaled to fit these.

The image type is specified by the extension of the given file name, so `.png` stands for PNG image, `tif` for TIF, and `.pdf` for PDF file. Otherwise the image is treated as JPEG (`jpg`).

6.4.2 `\pdfimageresolution` **=** *number*
We already mentioned the default resolution of 72 dots per inch. It is possible to overrule this value by using this register. Of course this only applies to bitmap PNG, TIF, and JPEG illustrations.

6.5 XObject Forms

The next three primitives support a PDF feature called 'object reuse' in PDFTEX. The idea is to create a Form object in PDF. The content of this XObject Form object corresponds to the content of a TEX box, which can also contain pictures and references to other XObject Form objects as well. After that the XObject Form can be used by simply referring to its object number. This feature can be useful for large documents with a lot of similar elements, as it can reduce the duplication of identical objects.

6.5.1 `\pdfform` *number*
Writes out the TEX box *number* as a XObject Form to the PDF file.

6.5.2 `\pdflastform`
Returns the object number of the last XObject Form written to the PDF file.

6.5.3 `\pdfrefform \name`
Inserts a reference to the XObject Form called `\name`.

As said, this feature can be used for reusing information. This mechanism also plays a role in typesetting fill–in form. Such widgets sometimes depends on visuals that show up on user request, but are hidden otherwise.

6.6 Annotations

PDF level 1.2 provides four basic kinds of annotations:

- hyperlinks, general navigation
- text clips (notes)
- movies
- sound fragments

The first type differs from the other three in that there is a designated area involved on which one can click, or when moved over some action occurs. PDFTEX is able to calculate this area, as we will see later. All annotations can be supported using the next two general annotation primitives.

6.6.1 `\pdfannot width` *dimension* `height` *dimension* `depth` *dimension* `{`*text*`}`
This primitive attaches an annotation at the current point in the text. The text is inserted as raw PDF code to the contents of annotation.

6.6.2 `\pdflastannot`
This primitive returns the object number of the last annotation created by `\pdfannot`. These two primitives allow users to create any annotation that cannot be created by `\pdfannotlink` (see below).

6.7 Destinations and links

The first type of annotation mentioned before, is implemented by three primitives. The first one is used to define a specific location as being referred to. This location is tied to the page, not the exact location on the page. The main reason for this is that PDF maintains a dedicated list of these annotations —and some more when optimized— for the sole purpose of speed.

6.7.1 `\pdfdest` ⟨`num` *n* | `name` *refname*⟩ `appearance`
This primitive establishes a destination for links and bookmark outlines; the link is identified by either a number or a symbolic name, and the way the viewer is to display the page must be specified; `appearance` must be one of those mentioned in table 3.

keyword	meaning
`fit`	fit the page in the window
`fith`	fit the width of the page
`fitv`	fit the height of the page
`fitb`	fit the 'Bounding Box' of the page
`fitbh`	fit the width of 'Bounding Box' of the page
`fitbv`	fit the height of 'Bounding Box' of the page
`xyz`	keep the current zoom factor

Table 3 The outline and destination appearances.

`xyz` can optionally be followed by `zoom` *factor* to provide a fixed zoom–in. The *factor* is like TEX magnification, i.e. 1000 is the 'normal' page view.

6.7.2 `\pdfannotlink width` *dimension* `height` *dimension* `depth` *dimension* `attr {`*attributes*`}` *action*
Starts a hypertext link; if the optional dimensions are not specified, they will be calculated from the box containing the link. The `{`*attributes*`}` are explained in great detail in the PDF Reference Manual and determine the appearance of the link. Typically, the attributes specify the color and thickness of any border around the link. Thus `/C [0.9 0 0] /Border [0 0 2]` specifies a color (in RGB) of dark red, and a border thickness of 2 points.

While all graphics and text in a PDF document have relative positions, annotations have internally hard–coded absolute positions. Again we're dealing with a speed optimization. The main disadvantage is that these annotations do not obey transformations issued by `\pdfliteral`'s

The *action* can do many things; some possibilities are:

`page` *n* jump to page *n*

`goto num` *n* jump to point *n*

`goto name` *refname* jump to a point established as *refname* with `\pdfdest`

`goto file` *filename* open a local file; this can be used with a `name` or `page` specification, to point to a specific location on the file

`thread num` *n* jump to thread identified by *n*

`thread name` *refname* jump to thread identified by *refname*

`user {`*specification*`}` perform a user–specified action; the PDF Reference Manual explains the possibilities; a typical use of this is to specify a URL, e.g. `/S /URI /URI (http://www.tug.org/)`

6.7.3 `\pdfendlink`

This primitive ends a link. All text between `\pdfannotlink` and `\pdfendlink` will be treated as part of this link. PDFTEX may break the result across lines (or pages), in which case it will make several links with the same content.

6.8 Bookmarks

6.8.1 `\pdfoutline` *action* `count` *n* `{`*text*`}`

This primitive creates an outline (or bookmark) entry. The first parameter specifies the action to be taken, and is the same as that allowed for `\pdfannotlink`. The `count` specifies the number of direct subentries under this entry; specify 0 or omit it if this entry has no subentries. If the number is negative, then all subentries will be closed and the absolute value of this number specifies the number of subentries. The `{`*text*`}` is what will be shown in the outline window (note that this is limited to characters in the PDF Document Encoding vector).

6.9 Article threads

6.9.1 `\pdfthread` ⟨`num` *n* | `name` *refname*⟩

Starts an article thread; the corresponding `\pdfendthread` must be in the box in the same depth as the box containing `\pdfthread`. All boxes in this depth level will be treated as part of this thread. An identifier (*n* or *refname*) must be specified; threads with same identifiers will be joined together.

6.9.2 `\pdfendthread`

Finishes the current thread.

6.9.3 `\pdfthreadhoffset` *dimension*

Specifies a threads horizontal margin.

6.9.4 \pdfthreadvoffset *dimension*
Specifies a threads vertical margin.

6.10 Miscellaneous

6.10.1 \pdfliteral {*pdf code*}
Like \special in normal TeX, this command inserts raw PDF code into the output. This allows support of color and text transformation. This primitive is heavily used in the METAPOST inclusion macros.

6.10.2 \pdfobj stream {*text*}
Similar to \pdfliteral, but the text is inserted as contents of an object. If the optional keyword stream is given then the contents will be inserted as a stream.

6.10.3 \pdflastobj
Returns the object number of the last object created by \pdfobj. These primitives provide a mechanism allowing insertion of a user–defined object into the PDF output.

6.10.4 \pdffontprefix {*string*}
In the PDF file produced by PDFTeX, one can recognize a font switch by the prefix F, for instance /F12 or /F54. This primitive can be used to force another prefix. This is only needed when one expects (or encounters) viewing problems with included PDF illustrations that use similar prefixes.

6.10.5 \pdfformprefix {*string*}
Forms are reusable graphic, textual or mixed objects. In the files made by PDFTeX such forms are internally identified by a number, which is not to be confused with the object reference as reported by \pdflastform. Like the previous and next primitive, this one can be used to overrule the default prefix, which is Fm, like in /Fm1.

6.10.6 \pdfimageprefix {*string*}
Like \pdffontprefix and \pdfformprefix, this primitive overrules a default prefix, this time Im, such as /Im58. Forget about these three primitives when you never encountered viewing problems, unless you want more fancy prefixes. When you do encounter PDF inclusion problems, change one or more of these prefixes in your document setup, and in the configuration file set include_form_resources to 1.

6.10.7 \pdftexversion
Returns the version of PDFTeX multiplied by 100, e.g. for version 0.13x it returns 13. This document is typeset with version 13.a.

6.10.8 \pdftexrevision
Returns the revision of PDFTeX, e.g. for version 0.13a it returns a.

7 Graphics and color

PDFTeX supports inclusion of pictures in PNG, JPEG, TIF and PDF format. The most common technique —the inclusion of EPS figures— is replaced by PDF inclusion. EPS files can be converted to PDF by GhostScript, Acrobat Distiller or other POSTSCRIPT–to–PDF convertors. The BoundingBox of a PDF file is taken from CropBox if available, otherwise from the MediaBox. To get the right BoundingBox from a EPS file, before converting to PDF, it is necessary to transform the EPS file so that the start point is at the (0,0) coordinate and the page size is set exactly corresponding to the BoundingBox.

A PERL script (EPSTOPDF) for this purpose has been written by Sebastian Rahtz. The TEXUTIL utility script that comes with CONTEXT can so a similar job. (Concerning this conversion, it handles complete directories, removes some garbage from files, takes precautions against duplicate conversion, etc.)

Other alternatives for graphics in PDFTEX are:

- LATEX picture mode
  Since this is implemented simply in terms of font characters, it works in exactly the same way as usual.
- Xy–pic
  If the POSTSCRIPT back–end is not requested, Xy-pic uses its own Type 1 fonts, and needs no special attention.
- tpic
  The 'tpic' `\special` commands (used in some macro packages) can be redefined to produce literal PDF, using some macros written by Hans Hagen.
- METAPOST
  Although the output of METAPOST is POSTSCRIPT, it is in a highly simplified form, and a METAPOST to PDF conversion (written by Hans Hagen and Tanmoy Bhattacharya) is implemented as a set of macros which reads METAPOST output and supports all of its features.
- PDF
  It is possible to insert arbitrary one–page–only PDF files, with their own fonts and graphics, into a document. The front page of this document is an example of such an insert, it is an one page document generated by PDFTEX.

For new work, the METAPOST route is highly recommended. For the future, Adobe has announced that they will define a specification for 'encapsulated PDF', and this should solve some of the present difficulties.

The inclusion of raw POSTSCRIPT commands —a technique utilized by for instance the `pstricks` package— cannot be supported. Although PDF is a direct descendant of POSTSCRIPT, it lacks any programming language commands, and cannot deal with arbitrary POSTSCRIPT.

Abbreviations

AFM	Adobe Font Metrics	MIKTEX	WIN32 distribution
AMIGA	Amiga hardware platform	PGC	PDF glyph container
AMIWEB2C	AMIGA distribution	PK	Packed Bitmap Font
CMACTEX	MACINTOSH WEB2C distribution	TETEX	UNIX WEB2C distribution
DJGPP	...	UNIX	Unix platform
EPSTOPDF	EPS to PDF conversion tool	WEB	literate programming environment
FPTEX	WIN32 WEB2C distribution	WEB2C	official multi–platform WEB environment
GNU	...		
JPEG	Joined Photographic Expert Group	WIN32	Microsoft Windows platform
MACINTOSH	MacIntosh hardware platform	ZIP	compressed file format

software

Introducing VTEX/Linux

Taco Hoekwater,
Bittext VOF
Michael Vulis,
The City College of New York & MicroPress, Inc.

abstract

This document is a short introduction to VTEX for Linux, a partial port of VTEX that is free for non-commercial use. The most interesting feature of the compiler is the use of PDF as a backend instead of DVI.

keywords

VTeX, Linux, port, MicroPress

What is VTEX/Linux

VTEX/Linux is a *partial* port of the VTEX/Win TEX compiler. It does not include the shell and/or Visual Tools and there is currently no intention to port those to Linux.

Even the port of the compiler itself is partial. Out of the three modes of VTEX/Win, only two are supported (PDF and DVI, but not HTML); the DVI mode is essentially useless, since the main advantages of VTEX's DVI mode under Windows rely on VTEX DVI drivers which are not being ported.

Thus, for all practical purposes, VTEX/Linux should be viewed as the PDF-mode compiler only.

Of the bitmap graphics filters that are supported by the Windows version of VTEX, only three have been ported to Linux at this time: the filters for PCX, TARGA and BMP files. Other filters that are used by the Windows version (currently consisting of filters for GIF, JPEG, PNG and TIFF) may be made available in the future.

On the other hand, VTEX/Linux includes the full PostScript support (GeX) of the Windows version. This includes both the EPS inclusion and inline PostScript, including support for PStricks, PSfrag, and GeXX.

The port requires Linux version 2.0, and at least 16Mb physical memory (without X11) or 32Mb (with X11). Below this limit performance will be unacceptably slow.

What is VTEX

VTEX is a full TEX GUI development environment for Windows. It features various visual tools to simplify input of formulas and LATEX pictures; a built-in editor with syntax highlighting and an integrated previewer; TEX DVI and PDF backends; an optional HTML backend; direct inclusion of EPS pictures and various bitmapped file formats; support for IF4, PFB, TTF and PK/MF fonts; virtual fonts; an in-line postscript interpreter (GeX); and a number of other extensions to the TEX language.

VTEX's current version number is 6.30. An (outdated) review of version 5.10 by Erik Frambach appeared in MAPS #20, page 142–145.

Direct PDF-generating mode

Starting at version 6.0, the VTEX Typesetter supports PDF output generating mode. Since PDF has become the defacto standard for publishing scientific documents online, this advance feature should prove of great benifit to users. Creation of PDF files from existing documents in TEX and LATEX is transparent. Various types of graphics, hyperlinks and outlines are fully supported.

Unlike other TEX systems supporting PDF output, VTEX builds the PDF directly from the TEX/LATEX source by the typesetter. There is no need for indirect conversion procedures like

$$\text{TEX} \rightarrow \text{DVI} \rightarrow \text{PostScript} \rightarrow \text{PDF}$$

or

$$\text{TEX} \rightarrow \text{DVI} \rightarrow \text{PDF}.$$

The typesetter incorporates Type1 and IF4 fonts with font subsetting (only the actually used characters are included in the PDF file). This generally results in compact and good PDF output. To ensure high quality of the produced files, the distribution comes with many standard TEX fonts in Type1 format.

GeX: Direct PostScript Graphics

Version 6.2 of VTEX introduces another major enhancement to TEX: an integrated PostScript processor/PDF translator. This extension (called GeX for 'Graphics eXtension') allows easy one-pass handling of Encapsulated PostScript files (`.eps`).

As of version 6.3, there is also direct support for the PStricks and PSfrag packages. There is more information on GeX in a separate article in this MAPS issue.

The most important feature of GeX is that in most cases there is nothing new to learn: GeX will take graphic[sx],

PStricks, PSfrag or seminar code without any changes. But there is more to GeX: with PostScript feedback, entirely new macro packages become possible.

Huge TEX

VTEX uses a HugeTEX version of the typesetter, which does away with many of traditional TeX limits. For example, the string, pool, and hyphenation space sizes are now limited only by the available memory. Perhaps the most irritating of TEX limits is the 256-font limit, which is no longer present: you can easily produce documents with thousands of fonts.

Installation and usage

A minimal working VTEX system is provided by MicroPress. It consists of several archive files. The web-pages mentioned below contain detailed up-to-date instructions on how to install VTEX/Linux.

A large portion of the VTEX distribution contains usual TEX files (for instance macros and font metrics). There are, however, some differences between the VTEX files and the 'standard' files, and you will be better off by using the MicroPress' supplied files.

Only the most essential components are duplicated to assure that the crucial packages function as tested by us. GeX relies on very recent corrections of PStricks, seminar and graphicx; these corrections may not yet be available in your TEX distribution. You can supplement the distribution with other standard packages available on CTAN as you desire.

General layout and configuration files

VTEX uses a file hierarchy that can be installed anywhere on your system. The default directory layout under Linux currently mimics the Windows version, with a number of subdirectories under one central directory called "vtex". It is very easy to adopt the system to your specific requests.

Unlike most other versions of TEX, VTEX does not rely on environment variables, but rather on a configuration file, `vtexlnx.rc` (in the user's home directory). This file uses a typical Windows `.ini` file syntax; it is divided onto several sections, each defining its variables. The only other needed configuration file is a font mapping file called `type1.rc`.

Both files are text; the exact description of these files needed for the customization is provided on the Web pages.

PDF Links and commands

One of the advantages of `.pdf` files is the ability to produce hyperlinks. VTEX supports both external and internal links. On the low level, this is accompished by VTEX `\special` commands. Several other commands are available to add information to the PDF document like Creation Date and Outlines. On the high level, all of the features are supported by S. Rahtz's `hyperref`; most features are also supported by a smaller and faster `pdf.sty`.

EPS file inclusion

The PDF backend supports `.eps` file inclusion. Prior to version 6.2, this inclusion was based on using GhostScript for some of the work. You had to install and set up GhostScript and ask VTEX to use it— otherwise the included file will be blank.

Starting at version 6.2, VTEX/Win includes the GeX converter which usually does a much cleaner job. VTEX/Linux does not support GhostScript piping at all since the use of GhostScript currently offers no advantages over GeX.

To activate `GeX` make sure to specify the `-ox` switch in the command line.

Unsupported VTEX syntax

Due to the limitations of the `.pdf` format, some VTEX extensions will not work. Specifically

- Grey rules are not supported (but colored rules using specials are).
- Font effects, except for `slant`, `aspect`, and the simplest form of `outline` are not supported on Type 1 fonts. All font effects are supported on `.if4` fonts.

License Terms

VTEX/Linux is currently available at no charge for *personal non-commercial use.* To use VTEX/Linux for any commercial purposes, you must obtain a commercial license from MicroPress.

At this time the software cannot be placed on any other server or on CD's.

Support and availability

VTEX/Linux is distributed on the internet from the following URL:

`http://www.micropress-inc.com/linux`

and by special permit also from the NTG's web server:

`http://www.ntg.nl/VTeX`.

The NTG has set up a mailing list where you can turn to for help and discussion. The list is called `ntg-vtex@ntg.nl`. You can subscribe to this list by sending a message to `majordomo@ntg.nl` with body "subscribe ntg-vtex".

Introducing GeX

Taco Hoekwater,
Bittext VOF
Michael Vulis,
The City College of New York & MicroPress, Inc.

abstract

This is a short introduction to the pilot release of GeX. GeX is the most rapidly evolving part of VTEX; more detailed documentation is available in the distribution of VTEX. This article specifically describes GeX as implemented in the public domain version of VTEX/Linux. While the same or additional features may be available in the commercial Windows version, we describe what exists in the freely downloadable version. For information on downloading VTEX/Linux see the NTG web site or the article in this MAPS issue.

keywords

pdf, inline graphics, eps inclusion, GeX, VTeX

Why GeX?

VTEX's PDF backend includes an integrated PostScript-compatible processor (GeX).

GeX makes it possible to do easy one-pass handling of:

- Encapsulated PostScript files (`.eps`)
- PStricks, PSFrag, and Seminar code
- Other inline PostScript code with optional feed-back of information from GeX to the TEX processor

While `.eps` inclusion has been previously supported in VTEX via GhostScript library calls, GeX offers much better performance and output quality.

The `.eps` inclusion is likely to be the main *initial* application of GeX. However, in our view it is the inline PostScript which could lead to new and interesting applications.

Why call it GeX?

The GeX name [pronounced `g-e-k-s`] stands for Graphics EXtensions.

While the current extensions are generally compatible with the PostScript language, GeX is intended to be a TEX-resident extension, not an Acrobat clone. Even in the current implementation there are facilities for communication between TEX and PostScript, as well as appoximately a dozen of new operators; these facilities are likely to be further developed. While it is our intention to stay PostScript-compatible to the degree needed for `.eps` and inline PostScript support, we envision further enhancing GeX with features that are decisively non-PostScript.

Enabling GeX

The current implementation does not enable GeX by default. This is because the initialization of the PostScript machinery takes 1-2 seconds and GeX is not needed for documents that contain only text and bitmapped images.

To enable GeX, use the `"-ox"` switch on the VTEX command line.

Notice that GeX works only in the PDF backend mode; in other modes the `"-ox"` switch has no effect.

Syntax

Supported PostScript operators

GeX currently supports a large subset of PostScript, including most of Level I and a few Level II operators. A full list of supported operators appears in the GeX reference documentation which is part of the distribution.

Since not the PostScript operator set is supported in its entirety, it is possible (and even easy) to write a valid PostScript code which will be rejected by GeX; on the other hand, the supported subset includes all the "practically" useful operators, so GeX would handle correctly essentially every `.eps` that appears in real life. Testing of GeX on a large random set of `.eps` files downloaded from the Internet shows that GeX correctly handles more than 99% of them.

Supported additional operators

GeX also understands a number of extra operators, of which the most important ones are listed below.

⟨**int**⟩ **.autofontload** If the integer argument is non-zero, GeX will query the `type1.rc` file when the `findfont` operator cannot resolve a font name. The default is *not to load* fonts implicitly and substitute Helvetica. This operator is useful for processing MetaPost-generated code; see the explanation at the end of this article.

⟨**string**⟩ **.loadfont** Loads a Type 1 font into the interpreter. The argument should be a string containing a font name. Only fonts listed in `type1.rc` can be loaded.

⟨**int**⟩ **.setdigits** This command sets the number of emitted fractional digits in the generated PDF output to an integer argument. The default value is two, which is the best value for most applications and devices. Only high-end applications may benefit from a larger value and only non-printable web-only files can do with a lower value.

.extend Will be explained below, in the section about extending GeX.

⟨**int**⟩ **.enabletransfer** A problem which arises with some `.eps` images is the use of the `settransfer` PostScript and related operators. The problem is that these operators are used for both device-dependant and device-independant color manipulations. The first usage is more common and is essentially for minor color adjustments. In such situations the best strategy for producing device-independant `.pdf` files is to disregard the transfer altogether. This is the default behaviour of GeX (and of the Acrobat Distiller).

However, in some (fortunately rare) `.eps` files the same operators are used to effect major device-independant adjustments. An example of such an adjustment would be the invertion of a black-and-white picture; this can be done with the

```
{ 1 exch sub } settransfer
```

PostScript code snippet. Disregarding this code will produce an inverted image. Thus, both Acrobat Distiller and GeX allow the user to process this inversion. In the case of Distiller, the override is a global Job option which will apply to all parts of a document; GeX allows one to override the handling of only an individual image. This is accomplished with the extention operator `.enabletransfer`. With an argument of zero, `.enabletransfer` disables processing of the `settransfer` code; a non-zero argument enables `settransfer` processing. Figure 1 is an example of a small `.eps` file that uses transfer code.

.tkwrite .tkread .tklength These operators are explained below.

Using GeX

If you are going to use GeX only for inclusion of ready-made `.eps` files, you should use a high-level package like `graphics` rather than VTEX's `\special`'s and disregard the rest of this section and paper. The same applies if you intend to use GeX with supported inline PS packages like PStricks, PSfrag or Seminar. In all these cases, the configuration files tuned up for GeX are supplied and knowledge of the low-level details in unneeded.

However, these details provide the framework for additional power to be realized in future packages for mixing text and graphics.

The GeX engine is invoked from VTEX with the following two `\special`'s:

- `\special{ps: ...}` is used to pass a file to GeX
- `\special{pS: ...}` is used to pass commands to GeX

With GeX enabled, VTEX allows you to precede a `\special` with the `\immediate` command. `\immediate \special`'s are passed to GeX right away, while TEX is still doing formatting.

The PDF code generated in *immediate* mode can be re-used, see the section below on re-using pdf code. Of course the immediate mode can also be used to do calculations.

TEX-GeX interface

The communication interface between TEX and GeX consists of three additional operators:

- `.tkread` to read the contents of a TEX `\toks` register
- `.tkwrite` to write to a TEX `\toks` register
- `.tklength` to find out the length of a TEX `\toks` register

The syntax of these operators is as follows:

- ⟨int⟩ ⟨string⟩ `.tkread` ⇒ ⟨int⟩ ⟨string⟩
  where the ⟨int⟩ parameter should be in the range 0 through 255 and designate a TEX token register; the ⟨string⟩ parameter is the receiving string. In the output, the integer value is the new length of the string; the string contains the contents of the `\toks` register.
- ⟨int⟩ `.tklength` ⇒ ⟨int⟩
  where the ⟨int⟩ parameter should be in the range 0 through 255 and designate a TEX token register; the output integer is the length of the contents of the TEX `\toks` register.
- ⟨boolean⟩ ⟨int⟩ ⟨string⟩ `.tkwrite` ⇒
  where the ⟨boolean⟩ argument defermines if the data should be appended to the `\toks` contents (`true`) or overwrite it (`false`); the ⟨int⟩ parameter should be in the range 0 through 255 and designate a TEX token register; the contents of the ⟨string⟩ parameter will be placed into the specified `\toks` register.

Note: During `.tkread` a `rangeerror` may occur if the `\toks` register contains more characters than can be placed into the receiving string; one can use the `.tklength` opera-

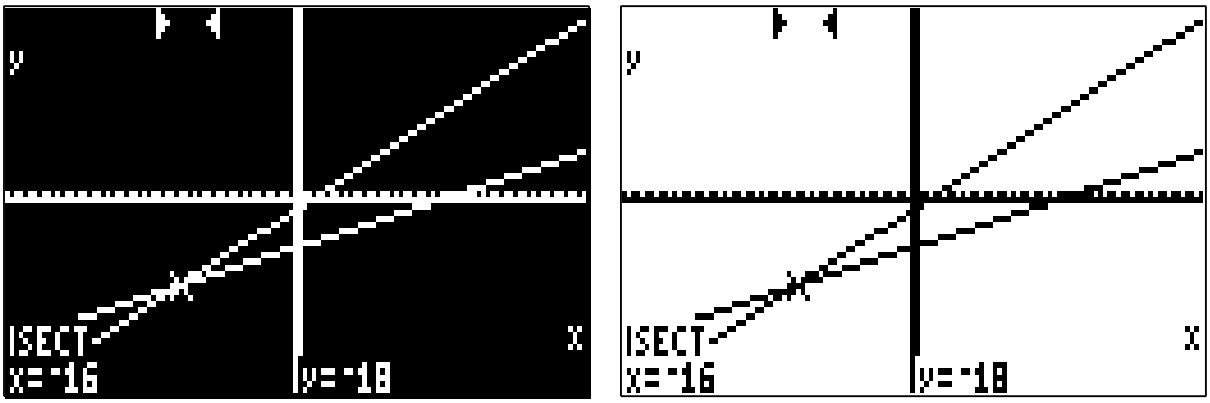


Figure 1. On the left the figure included with default settings. On the right the figure as it would appear after enabling `settransfer`.

tor to find out how big the receiving string should be before allocating it.

Note: Control sequence tokens withing TEX token strings are converted into spaces during `.tkread`; they are counted as single characters in `.tklength`.

Note: Token strings produced by `.tkwrite` contain only tokens with TEX `\catcode` 12 (other).

Re-using pdf code

VTEX allows you to re-use the pdf code that is generated by the `\immediate` form of the `\special{pS:...}` operator.

During the `\immediate` output, the pdf code is written to a temporary stream. Any `\immediate\special{pS:...}` operator opens such a stream (unless it is already opened by another operator); the currently opened stream, if it exists, is destroyed at the moment of `shipout`. Thus, by default, everything written to immediate streams is lost.

To preserve the contents of an `immediate` stream, use the `\special{ice}` command. This command closes the immediate stream; the new TEX count register `\pdflaststream` can be used to retrieve the handle to the closed stream that was just closed. The command

```
\special{!stream \the\pdflaststream}
```

can be used to re-insert the frozen stream into the TEX machinery; it will be emitted during the normal shipout.

Notice that the `\special{ice}` command must be issued in the `\immediate` mode (otherwise, there will be no stream to freeze by the time it gets processed); on the other hand, `\special{!stream ...}` must be deterred till the `\shipout`.

If you generate pdf code during the `\immediate` mode, you should realize that the positioning of your code will not be known until the time of the `\shipout`. Thus, the PostScript `currentpoint` is not really defined. The way to overcome this problem is to initialize the current point to $(0,0)$ by executing `0 0 moveto` at the beginning of the `\immediate` stream.

The data inserted in the output during the `\special{!stream...}` processing is offset by the `currentpoint` as computed during the `\shipout`.

Extending GeX

The major new feature added in VTEX 6.3 is GeXX. The second "X" stands for eXtensible. With GeXX you can supplement the existing set of operators with new constructs, implemented in C or C++.

In the Linux version, the extension libraries are standard shared objects (.so), implemented with gcc/g++. While the GeX API seems C++ at first sight, it is actually standard C. The GeX API is portable, so the same extension can be compiled for both Linux and Windows (under Windows it would become a .dll). At this moment the only supported compiler under Windows is BC, but it is likely that other compilers and languages can be used on both platforms.

The libraries are not referenced from the executable, so you can create new ones as you desire; a single library can implement multiple extentions, and multiple libraries can be loaded in the same job.

To enrich GeX with new operators, you should

- Implement them within a C-language library
- Call the `.extend` operator to load the new language extensions
- Provide additional TEX and/or PostScript code for easy access to the new operators

Note: To be visible to the compiler, the extension library should be placed into the `vtex/bin/gex` subdirectory.

An extention library is only required to export three functions:

1. A function that returns the version of the interface as defined in `gexi.h` (If the version returned by the extention library does not match the version needed for the VTEX compiler, the library is not loaded)
2. A function that returns the number of extensions that are implemented
3. A function that returns the PostScript names of the new extensions and a pointer to the function that should be called when a specific extension is encountered

If you decide to make your extentions publicly available, you should make sure that it is clear for which version of VTEX the extention is designed. This means either supplying the extention in the source form (recommended), or at least mentioning the version number in a readme file. Public distributions of VTEX/Linux will be glad to host your extentions.

Writing an extention operator

An extension operator should be declared as an `int` function; its solo argument is the GeX interface structure, `GEXI`.

The majority of the methods provided by `GEXI` correspond one-to-one to either PostScript operators with the same names, or GeX extention operators (`.tkread`, for example). The only exceptions to this at the present time are the methods that deal with the PostScript operand stack; these methods are used to retrieve (and, later, pop) the arguments provided on the operand stack.

Loading an extension library

To load an extension library, execute the `.extend` operator.

The syntax is: ⟨string⟩ `.extend` ⇒ ⟨int⟩

where the ⟨string⟩ argument contains the name of the library file with the language extentions (without the file extension) and the returned ⟨int⟩ is the number of extention operators loaded.

The `.extend` operator will fail with an error if:

- the specified DLL cannot be found
- the specified DLL does not export all three required functions (`Count()`, `Version()`, `Names()`)
- the version returned by the DLL does not match the version of the VTEX compiler

In all three cases, `.extend` will cause a PostScript `fileerror`.

PieChart

PieChart is a real-life example of using GeXX, which implements MS Word-like PieCharts in TEX. The implementation consists of

- `piechart.[dll|so]`, the extension library.
- `piechart.Sty`, a LATEX2E style for using PieChart.

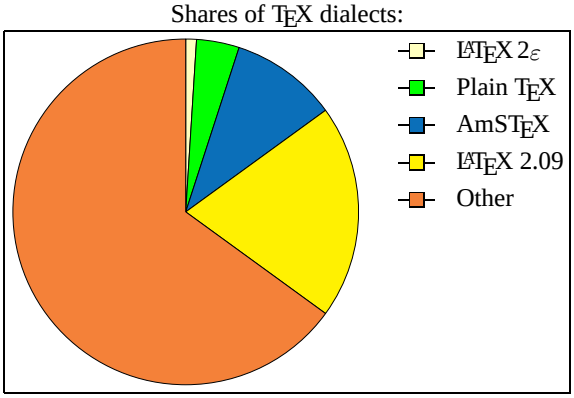
Here is some sample code using PieChart:

```
%% Define some colors
\definecolor{lightyell}{rgb}{1,1,0.75}
\definecolor{peach}{cmyk}{0,0.50,0.70,0}
\definecolor{orange}{cmyk}{0,0.61,0.87,0}
\definecolor{navyblue}{cmyk}{0.94,0.54,0,0}

\begin{center}
Shares of \TeX\ dialects:\par
\fbox{\begin{PieChart}[rt]{1.8in}
\PieSlice{orange}{65}{Other}
\PieSlice{yellow}{20}{\LaTeX\ 2.09}
\PieSlice{navyblue}{10}{AmS\TeX}
\PieSlice{green}{4}{Plain \TeX}
\PieSlice{lightyell}{1}{\LaTeXe}
\end{PieChart}}
\end{center}
%%
```

and a sample PieChart produced by this extension:

Shares of TEX dialects:



The PieChart package has been written by Alex Kostin at MicroPress.

Bugs

Being a pilot implementation with source of about 15000 lines of code, GeX undoubtly has many bugs. More than a hundred of them were fixed since the happy moment in July when we thought it more-or-less worked (and were proven wrong on testing of huge set of *real-life* `.eps`'s from diverse sources).

In the aggravation of fixing what we thought was a working program, we discovered that the bugs came in three flavors:

- Our bugs
- Peculiarities (often undocumented) of the PostScript language
- Bugs (or problems) in Adobe Software

Bugs of our implementation (important for us for sentimental reasons) are not worth discussing here; but some of the other bugs are definitly worthwhile mentioning.

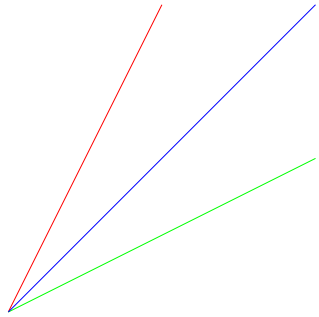
Degenerate matrices

Near-degenerate matrix transforms cause a serious problem with the Acrobat's 16-bit computational limit. It can be shown that the problem is not solvable correctly in general; and Adobe Acrobat Distiller fails on degenerate transforms.

The example file

```
% lwid.ps
0 0 moveto
gsave 100 200 lineto 2 3 scale 1 0 0
                  setrgbcolor stroke grestore
gsave 200 100 lineto 0.5 0.3 scale 0 1 0
                  setrgbcolor stroke grestore
gsave 200 200 lineto 0 0 1 setrgbcolor
   [0.186718 -0.565306 0.873838 -2.64563 0 0]
                                setmatrix
   stroke grestore
showpage
```

should produce three lines from the origin. Distiller, however, will miss the middle line. GeX, on the other hand, will produce correct output:



Near-degenerate matrices are not a perverted abberation: they tend to be generated by some common software, especially CorelDraw. The particular set of numbers in the source above came from a Corel example.

While GeX does the work correctly in all cases, some distortion in the line widths is possible and is not avoidable.

Level 1 strokeadjust

Some graphics programs (Freehand is one) output Level I PostScript code which fits the coordinates to an integer grid. This code, if executed literally, will produce rather disasterous results with GeX.

The nature of the problem is a bug (or *feature*) in the Freehand adjustment code which does not bother to check for the device matrix and assumes that it corresponds to the output pixel resolution of 300 dpi or higher (which would imply a device matrix `[4 0 0 4.. ..]`). However, the GeX device matrix is chosen to be an identity, to avoid extra rounding by TEX's ⇔ GeX's coordinate translation. This causes extremely coarse coordinate rounding (72dpi) in the default case.

An example of this effect is provided in the VTEX/Linux distribution.

Font name collision bug

There seems to be a bug in many versions of Acrobat which results in (different) fonts with names starting with `|------...` being treated as a single font. To avoid this problem, we replace such names with `|xxxxxx....`

Encoding bug

Under Windows, the Acrobat Reader seems to ignore the `/StandardEncoding` specification and uses the `WinAnsiEncoding` instead. This may lead to incorrect character substitution for some codes in the 2nd half of the ASCII set.

To overcome this problem, VTEX always includes the encoding vector, even if the font is not reencoded.

Dirty Tricks and examples

show redefinition

In order to accomodate packages such as PStricks and PSfrag, VTEX keeps track of redefinition of the `show` PostScript primitive within the GeX engine. In addition to supporting the mentioned packages, this allows rather nice font effects to be implemented with very simple inline code.

Simple outline The examples below were produced with

```
\def\outl#1{\special{pS: save /show{false
   charpath stroke}def}#1\special{pS: restore}}
```

This is a test.

Wider outline with color The macro

```
\def\outla#1{\special{pS: save /show{false
```

```
  3 setlinewidth 1 0 0 setrgbcolor charpath
  stroke}def} #1\special{pS: restore}}
```

produces

Filled letter with outline

```
\def\outlb#1{\special{pS: save /show{false
 charpath gsave 2 setlinewidth 1 0 0 setrgbcolor
 stroke grestore 0 1 0 setrgbcolor fill}def}
 #1\special{pS: restore}}
```

produces

Charpath shown

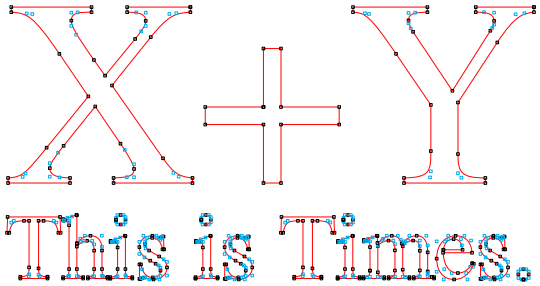
We can also get inside the character representation (something which PostScript would not do on Type 1 fonts):

```
\def\outlc#1{\special{pS: save

  /rct{
   newpath 0.5 add exch 0.5 add exch moveto
   currentpoint exch -1 add exch lineto
   currentpoint -1 add lineto
   currentpoint exch 1 add exch lineto
   currentpoint 1 add lineto stroke} def

  /show{
  false charpath gsave
  0 setlinewidth 1 0 0 setrgbcolor
  stroke grestore
  0 setlinewidth 0 0 0 setrgbcolor
  {rct}
  {rct}
  {rct 1 0 0 0 setcmykcolor
  rct rct 0 0 0 setrgbcolor}
  {} pathforall }def}
  #1\special{pS: restore}}
```

to obtain:



Fragment repositioning

Several examples in PSTricks use the PostScript commands to move the text around in order to land it in an appropriate place on a drawing.

VTEX keeps track of PostScript attempts to group the TEX output; when such activity is detected, VTEX generates PostScript code rather than PDF and feeds this code into the GeX engine.

MetaPost support

While GeX can handle MetaPost-generated files, it is important to state that MetaPost outputs invalid EPS files. Rather than use the standard fonts or embed fonts in EPS, MetaPost merely includes declarations like:

```
/cmr10 /cmr10 def
```

and expects post-processing to find and substitute the fonts. Instead of such post-postprocessing, GeX ignores (processes, which is the same really) this declaration, but requires either explicit loading of needed fonts via the `.loadfont` extension:

```
\special{pS: /cmr10 .loadfont}
```

(one such command for each required font) or enabling of the autoloading feature via the `.autofontload` extension

```
\special{pS: 1 .autofontload}
```

These commands must be issued before a MetaPost-generated file is actually included.

Acknowledgements

The authors wish to express thanks to:

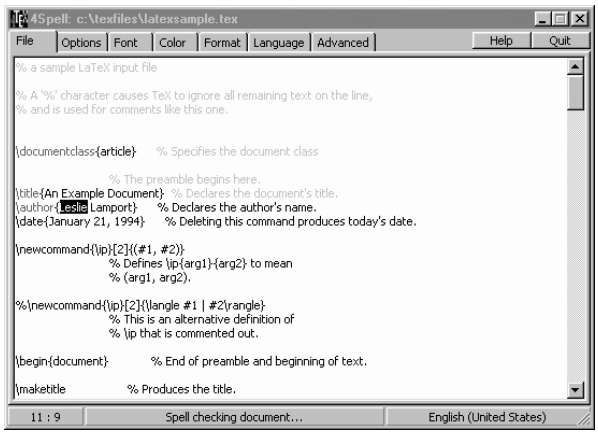
- Alex Kostin for extremely heavy testing of preliminary versions of GeX and finding a few dozen glitches.
- Denis Girou and Timothy van Zandt for cooperation and help in cleaning bugs in PStricks and Seminar which made their use with GeX possible.

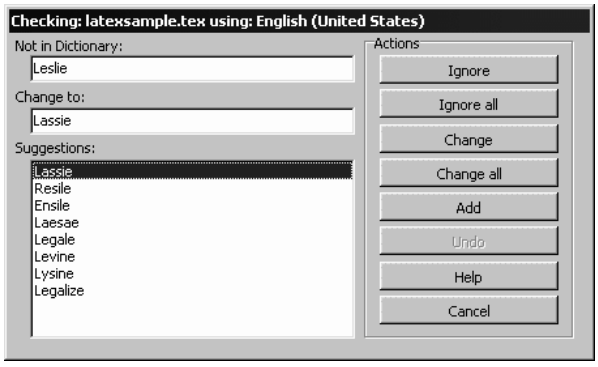
4Spell, a spell-checker for Windows 95/98/NT

Wietse Dol and Erik Frambach

abstract

In this paper we will describe the features of 4Spell 1.1, a Windows spell-checker for TEX documents. Since there aren't many good spell-checkers around and since 4Spell only works on Windows platforms, we will also explain how the spell-checking is done. This should make it possible to write a spell-checker for other platforms (why not use perl and become platform independent :-) 4Spell is part of the new 4TEX for Windows (release expected by the end of March 1999). We realized, however, that this tool could be useful for people who do not want to use 4TEX and hence we made it a stand-alone freeware program.





Introduction

Spell-checkers are nowadays widely used by word processors such as MS-Word and WordPerfect. They are extremely useful in correcting spelling errors, especially when writing in a non-native language.

TEX users often claim that TEX is better than those word processors, one wonders why there are so few good spell-checkers for TEX documents.

The main reason for this is that TEX documents not only contain "normal" words, but also complex TEX commands. And TEX commands may or may not take parameters, and parameters can be delimited in any imaginable way.

Within certain TEX environments you want the words to be checked (e.g. in tables) and in others you want them to be ignored (e.g. mathematics). All in all a complex situation if you realize that TEX commands, mathematics, and normal words needn't be separated by spaces and line feeds. Any good spell-checker for TEX documents requires a TEX parser that reads the text and decides whether or not a word or a part of the word should be spell-checked. Is writing a parser difficult? The answer probably would be "yes", since there aren't many spell-checkers around. 4Spell proofs that writing such a spell-checker can be done and that it's not that hard to write a spell-checker that can even do more than just TEX.

4Spell features

When we started to write 4TEX for Windows we still needed the "old" MS-Dos based AmSpell as a spell-checker. AmSpell has some serious problems/bugs when it checks your documents. We will not give you a list of those problems, but after AmSpell has checked your document you still can find spell-checking errors. This is because Amspell skips parts of your document and doesn't tell you it did.

In September 1998 we had the discussion if we needed to write a spell-checker for 4TEX and concluded that it should be too time consuming to write a good program, since TEX documents are too complex. As often, complex material tends to become much simpler when you have a closer look and spend more time thinking about the structures (TEX is a structured language isn't it). When starting to write 4Spell we started not on the spell-checking routines but on describing how a TEX document should be parsed through the spell-checker. This parsing is the engine of a good spell-checker (and here AmSpell makes it's

mistakes). The spell-checking routines were supplied by Aleksander Simonic. Alex is the author of WinEdt, probably the best TEX-aware shareware editor there is for OS/2 and Windows. Cooperation with Alex means that we can all benefit from the same dictionaries, which makes maintenance a lot easier.

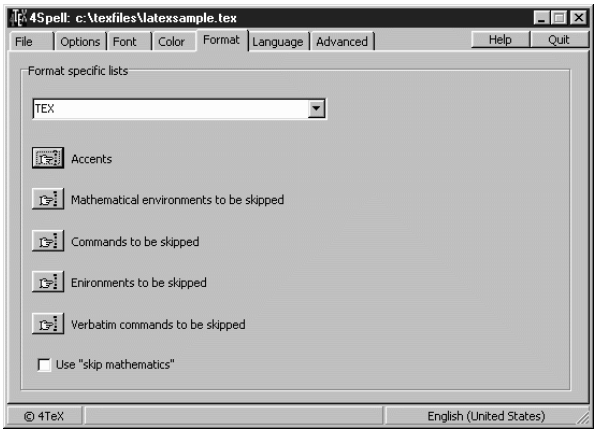
In the next section we will describe the parsing of a document, but now we will summarize some of 4Spell's features:

- Color highlighting, i.e. actions of the spell-checker are translated into coloring of words. This makes it easy to see how your document was interpreted and (possibly) changed. Not useful you would say!? But we discovered it is extremely powerful. For instance suppose you want mathematics to be skipped by the spell-checker and you have written in your document

  ```
  This example $x+y will trigger probblems
  ```

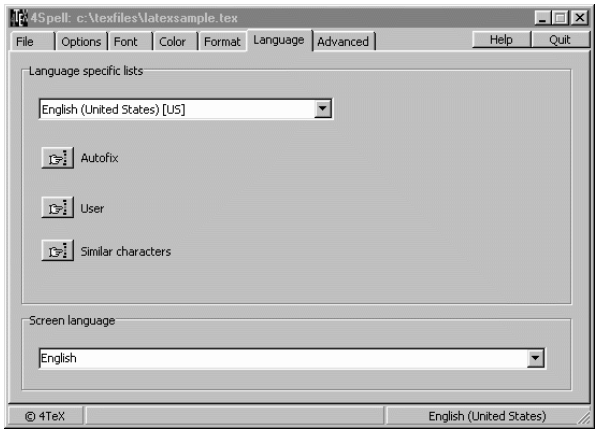
  Can you predict what will happen if you check your document: it will skip the whole document after the `$xy+` since the mathematics isn't ended properly. With AmSpell (or any other spell-checker) you couldn't see this. Now you can see and solve the problem just by looking at the colored document (i.e. everything after the mathematics statement `$xy+` is colored as mathematics)!
- Support of many languages: English (American and British), German, Dutch, French, Italian, Swedish, Danish, Russian, Polish, Spanish and South-African. Since all dictionaries are plain ASCII it is simple to add your own language or to update one of the dictionaries.
- Language switching within a document allows you to write multilingual documents and spell-check all parts according to the language in which they were written.
- Generation of a word list. All the different words that are used within your document can be listed. This can be useful in deciding which words should be considered for indexing.
- Generation of a logfile, showing all actions and changes that were performed.
- Basic statistics of the document and the spell-checking run are recorded.
- Many options that can be switched on/off to increase speed and/or performance.
- You can select fonts, font sizes and character sets. This makes it possible to spell-check non-western documents (Polish, Russian, etc).
- All colors used can be changed according to your personal preferences.
- 4Spell is format dependent. It maintains specific lists for each format that you use. Lists are defined for: accents, mathematical environments, commands, environments and verbatim commands. Also mathematics (between $...$ or $$...$$) can be ignored. Note that 4Spell supports (by default) the following formats:
  – TEX and LATEX documents
  – Rich Text Format (RTF) documents
  – plain ASCII documents
  – HTML documents
  – BibTEX documents

  Indeed, not only TEX documents can be checked and hence makes 4Spell useful not only for TEX users. It is easy to add a format (e.g. ConTEXt) and make the chenges to one of the format dependent lists and properties.



- 4Spell is language dependent. It maintains specific lists for each language that you use. Lists are defined for: automatic correction of typing mistakes, user specific words, and similar characters (used to specify which letters/characters are associated when looking for alternatives of an incorrect word).
- 4Spell can change its user-interface language on the fly (as 4TEX and 4Project).
- All settings for words, subwords, punctuation marks, etc., are format specific (see also the next section). This makes it easy to spell-check not only documents written in TeX, but also Plain ASCII, HTML, or RTF files. And of course you can define you own formats.
- You can check wether a word is used twice. For instance have a look at the word "one" in the next example:

  ```
  When you write very long lines and you end
  end one with a small word you tend to write
  certain words twice.
  ```



4Spell will ask you if you want to delete the second ”end” entry.

- 4Spell is lower and uppercase sensitive. For instance words as ”This”, ”tHis” and ”THis” can be changed automatically in ”This”. When checked by 4Spell it will give the suggestion ”This” for the words used in this example.

The parser

All functionality above is mostly the result of writing a (TEX) parser. To make it easier for others to write their own parser, and for those who are just curious to know how it works, we will explain the parsing algorithms.

The parser will read words until the end of a file is reached. This is done by letting a pointer start at the beginning of the file and start with the procedure `READWORD`.

STEP 1: get a word procedure READWORD

1. Skip `EndOfWord` characters until the first non-`EndOfWord` character.
2. Read and remember characters until the first `EndOfWord` character.

The result of 1 and 2. is a **word**.

This `READWORD` procedure is repeated until the end of the file. With these words you need to do a lot of checks before you can spell-check (since a word as defined above can contain (TEX) commands, etc.). Note also (within TEX) the `EndOfWord` characters are defined as: a space, a hyphen, a tilde, a Carriage-Return, a Line-Feed, and an End-Of-File character.

STEP 2: check the word for properties

For every word check the following:

1. check if the `Language Switch` (i.e., the command that is used to change dictionaries) is part of the word
2. check if one of the commands in the (TEX) `Begin Environments list` is part of the word
3. check if one of the commands in the (TEX) `commands list` is part of the word
4. check if one of the commands in the `Begin Mathematics Environments list` is part of the word
5. check if the `Mathematics Command` (e.g., $x+y$ or $$x+y$$) is part of the word
6. check if the `Verbatim command` is part of the word
7. check if part of the word starts a (TEX) comment (i.e. the % sign)

If one of the above is true you keep on reading words until:

1. the characters after the Language Switch command is the filename of the dictionary that should be loaded at that point.
2. the `End Environment command` is part of the word
3. the `End command command` is part of the word
4. the `End Mathematics Environment command` is part of the word
5. the `End Mathematics` is part of the word
6. the `End Verbatim character` is reached
7. end of the line is reached

This seems easy, but the problem is that when looking for, say, an environment to be ended, the same environment can start again and hence we do not stop at the first end-environment part, but at the second (or even higher) end-environment parts. This example will hopefully explain the problem:

```
\begin{skipping}
  To explain the spell problem see this example
  \begin{skipping}
    This won't work if you do not count the number
    of begin environments
  \end{skipping}
  You understand the example?
\end{skipping}
```

What the spell-checker should do is skip the complete example above. It should not stop skipping at the first `\end{skipping}` command.

When performing the actions above, we were looking for parts of the words. This means that after these actions we will have found (part of) a word preceding the action

and (part of) a word after ending the action. With these two words (which may be empty) we proceed as with a word that doesn't trigger one of the actions described above.

STEP 3: divide word into subwords
Look if the word contains `SubWordPunctiationMarks`. If so, divide word into subwords.
`SubWordPunctuationMarks` are

```
,;:.!@#$&*?"%(){}[]-+=0123456789\'~^*_/|'
```

An example could clarify the meaning of subwords. Suppose we have the word

```
\def\hello{\textbf{Hello}}
```

This will be divided into four subwords:

```
\def
\hello
\textbf
Hello
```

STEP 4: check the subwords for properties
These subwords are candidates for spell-checking, but befor we spell-check these subwords we check:

1. Does the subword start with a (TEX) `Command Character` ("\"), then skip the subword (so the first three subwords of the example are skipped).
2. Is the subword one of the words in the `Ignore words list`, then ignore it.
3. Is the subword one of the words in the `Replace words list`, then replace it automatically.
4. If the subword is one of the words in the `User Dictionary`, then skip the subword.
5. If the subword is one of the words in the `Ignore Dictionary`, then skip the subword.
6. If the subword is one of the `Auto Replace word list`, then replace the subword with correct word from the `Auto Replace with word list`

If the subword doesn't belong to any of the six categories above, we spell-check the subword (Alex's routines do the job fast and easy). If the subword is correct we skip the subword. If it is not a correct word, we will search for alternatives for this (sub)word. The user will be prompted by 4Spell what to do in this case: select one of the alternatives, enter your own text, ignore this word, or add it to the user dictionary.

It seems easy, but be aware that when building a parser, you will you need to do a lot of bookkeeping, and you will need some more advanced programming tricks (e.g., all word actions and subword actions are recursive procedures).

software

4Project: a project manager for TEX

Erik Frambach
Rijksuniversiteit Groningen
email: E.H.M.Frambach@eco.rug.nl

abstract

A new project manager, 4Project, analyses your TEX document and gives you easy access to any item in it.

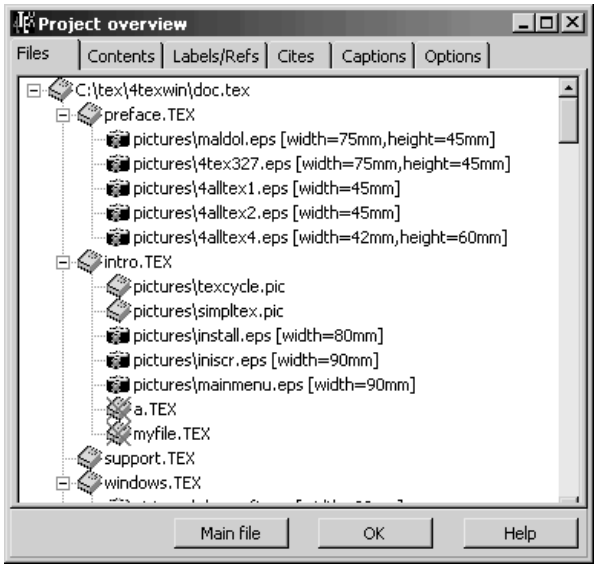
keywords

project management

Introduction

When writing large documents that contain many chapters, sections, figures, tables, references, citations, etc., it can be difficult to keep track of the overall structure. Furthermore, large documents are often divided into multiple files, possibly written by different authors.

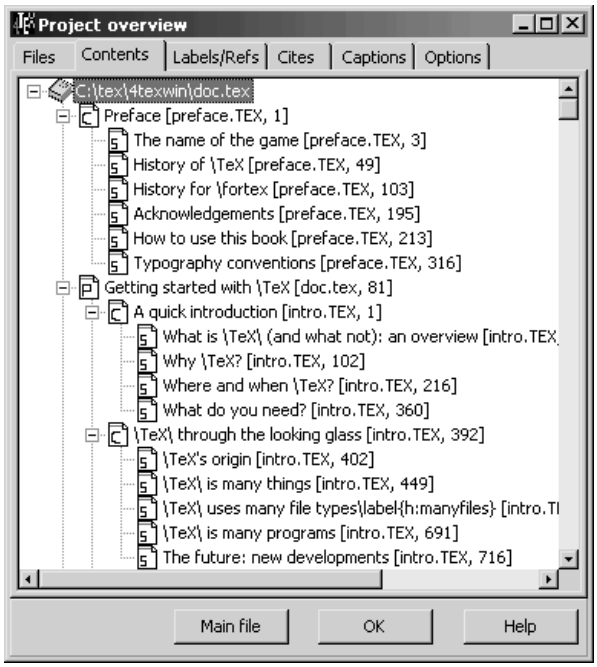
In such cases a good project manager can be helpful. It will assist you in analyzing the structure and content of a document, and it will give you easy access to any part of the document, be it a file, a section, a graphic, a label, a cite, or whatever. It will also show potential problems such as resources (files, graphics, etc.) that it was unable to locate, references to undefined labels, multiple defined labels, labels that were never referenced, inconsistent sectioning, etc. A graphic representation of these analyses will give you a detailed yet easy to manage overview of your document.

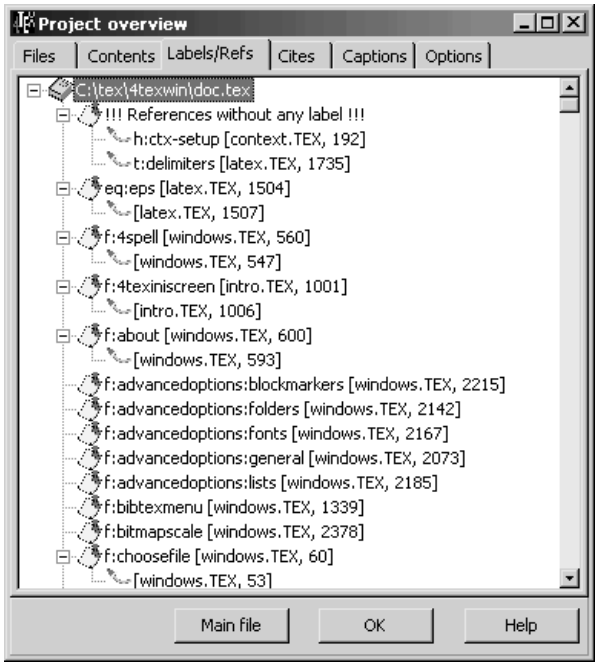


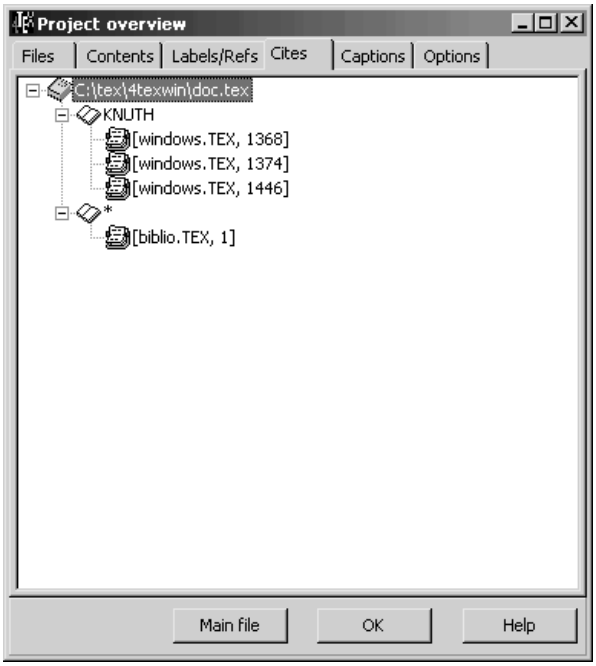
4Project

The program 4Project is an attempt to implement such a project manager for MS-Windows TEX users. The program is part of 4TEX version 5 for Windows, but it can be used 'stand-alone' just as well. 4Project is highly configurable, which is necessary to make it work with, e.g., LATEX, ConTEXt, plain TEX, or whatever dialect you use. These dialects all use different commands to include files, graphics, citations, etc., that 4Project needs to know in order to analyze such a document.

4Project will display the results of its analyses in 5 sections that we will discuss below.







Files analysis

The first 'tab sheet' in 4Project is called 'Files'. It displays a tree of all the TEX files that the document includes through, e.g., `\include` commands. It also shows which external graphics files are included.

Any file that could not be found will be indicated by a red cross through its icon. If you click on a file displayed in this tree, 4Project will start the editor (which you can specify) and load this file. If you click on a graphic file, that file will be displayed by a suitable graphics program such as GSview or IrfanView (which is of course configurable). If you *right*-click on a file, the program 4Spell will be started to spell-check that file.

Contents analysis

The 'Contents' tab sheet displays a table of contents in which chapters, sections, subsections and subsubsections are listed. File names and line numbers where these items are defined are also displayed.

By clicking on an item, the editor will be started, and the file will and the editor will jump to the given line number.

Labels/Refs analysis

The 'Labels/Refs' tab sheet displays all labels and references used in the document.

If there are any references to undefined labels, they will appear at the top. Similar to the contents analysis, all file names and line numbers are given as well. By clicking on an item the editor is started. By right-clicking on an item, the label name will be copied to the Windows clipboard. From the editor you can then paste it into a file.

Cites analysis

The 'Cites' tab sheet displays all citations in the document.

By clicking on an item the editor will be started. By right-clicking on an item, the citation name will be copied to the Windows clipboard, so you can paste into a document from the editor.

Captions analysis

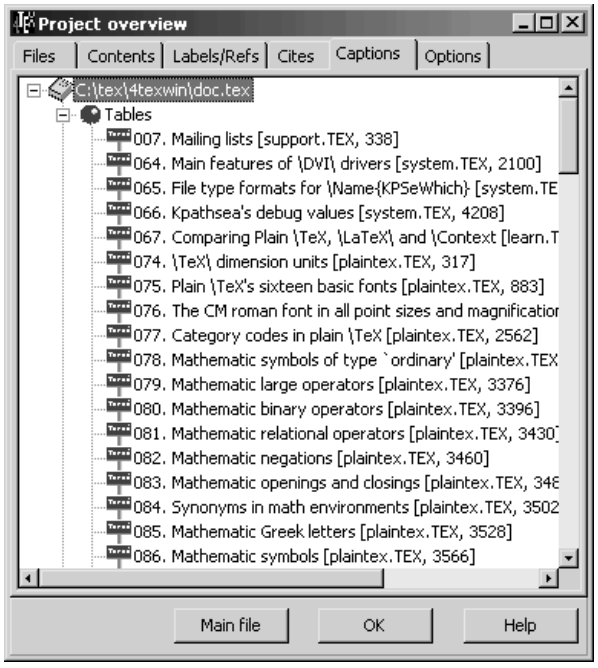
The 'Captions' tab sheet displays all captions used in the document. The tree view is much like a list of figures and list of tables.

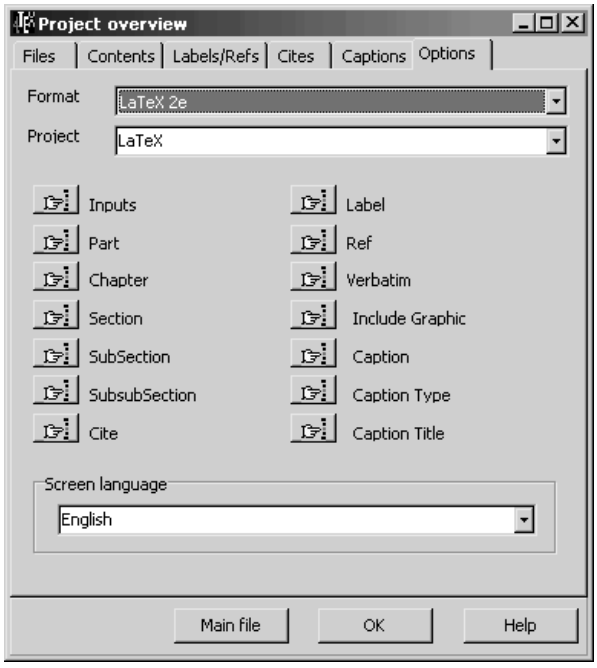
By clicking on an item the editor will be started.

Options

4Project is highly configurable. The 'Options' tab sheet gives you easy access to all major parameters.

You can specify what commands 4Project should scan for to find, e.g., input files, chapters, references, figures.





As an example, here is a definition for input files in LaTeX:

```
\input
\include
```

A definition for figure/table captions in ConTEXt could look like this:

```
\placefigure
\startfiguretext
\placetable
```

All these settings and some more are stored in the file `4tex.ini`.

4Project is free software, which means that you don't have to pay for using it. The standard GNU license applies. You can download it from any CTAN (mirror) site (e.g., `ftp.ntg.nl`). It is part of 4TEX, (`systems/win32/4tex/`) but you can throw away anything you don't want. Later, 4Project will be made available as a separate distribution on CTAN.

perl scripting

How Perl can help TEX

Wybo Dekker
wybo@servaly.hobby.nl

abstract

Perl may be an easy interface to TEX when it comes to repetitive tasks, like writing letters, creating reports from databases, and many more. This article shows how Perl can be used to generate many similar pictures *via* the MFPIC style

keywords
perl, mfpic, mkpic

1 Introduction

I recently had to produce about 40 pictures for insertion into a book on elementary mathematics. I decided that the MFPIC would suite most of my needs. But writing MFPIC commands is not easy. Figure 1, for example, can be constructed using the following MFPIC commands:

```
1  \mftitle{ce}
   \setlength{\mfpicunit}{1mm}
   \begin{mfpic}[16][5.45]{0}{4}{-6}{5}
   \axes
5  \hatchwd{2}
   \tlabel[bc](0,5.54){$y$}
   \tlabel[cl](4.21,0){$x$}
   \tlabel[tc](2,-0.18){\strut 2}
   \tlabel[bc](3,0.18){\strut 3}
10 \tlabel[cr](-0.07,-5){\strut -5}
   \tlabel[cr](-0.07,0){\strut 0}
   \tlabel[cr](-0.07,4){\strut 4}
   \rhatch\lclosed\connect
   \lines{(0,0),(0,4)}
15 \function{0,3,.05}{4-x*x}
   \lines{(3,-5),(3,0)}
   \endconnect
   \function{0,3.2,.05}{4-x*x}
   \dotted\arrow\lines{(3,-5),(0,-5)}
20 \dotted\arrow\lines{(3,-5),(3,0)}
   \tlabel[bc](3,4){\parbox[b]{60mm}{%
     \center $f(x)=4-x^2$}}
   \arrow\lines{(3,3.46),(1.7,1.1)}
   \tlabel[bc](2,5){\parbox[b]{60mm}{%
25   \center Area $O_1$}}
   \arrow\lines{(2,4.46),(1,2)}
   \tlabel[bc](4,2){\parbox[b]{60mm}{%
     \center Area $O_2$}}
   \arrow\lines{(4,1.46),(2.8,-2)}
30 \end{mfpic}
```

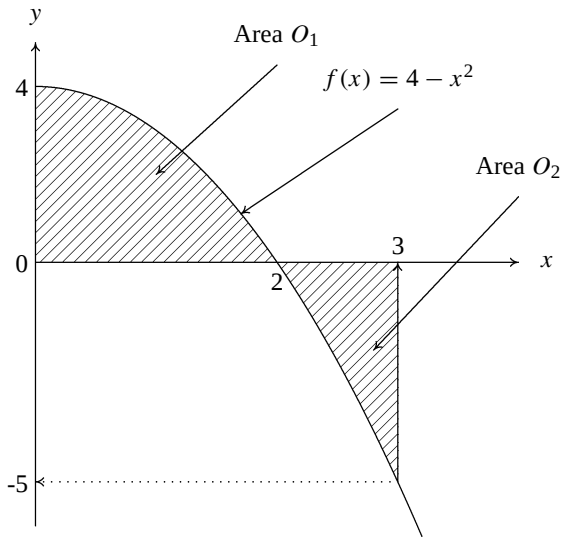


Figure 1. example a

As you can see, this implies a lot of typing and one has to type many nested [], {}, and () pairs. Also, several floating point numbers, such as those in lines 6–12, depend on the scaling factors defined in line 3. They have to be calculated manually, and changing the scale will imply recalculation of those values. The scale itself is set in line 3: I wanted the picture to be 64 mm wide, so I had to calculate $64/(4-0) = 16$ for the scaling factor in the x-direction. It would be much easier if one could type something like:

```
1  begin ce 64 64 0 -6 4 5 $x$ $y$
   xmark 2
   Xmark 3
   ymark -5 0 4
5  bhat
   lines 0 0 0 4
   func 0 3 .05 4-x*x
   lines 3 -5 3 0
   ehat
10 func 0 3.2 .05 4-x*x
   xydrop 3 -5
   arrow 3 4 1.7 1.1 $f(x)=4-x^2$
   arrow 2 5 1 2 Area $O_1$
   arrow 4 2 2.8 -2 Area $O_2$
15 end
```

Here we see no brackets, braces or parentheses anymore, width and height are set straightforwardly to 64 mm and the labels along the axes are redefined as xmarks and ymarks,

for which nothing has to be given but the x- and y-values, respectively. The corresponding y- and x-values are supposed to be calculated automatically.

Another construction that frequently occurs in my pictures is a label with an arrow starting from the center of its baseline, such as the one in lines 21–23 in the long listing. This is replaced in the short listing with line 12, where the starting position of the arrow is supposed to be calculated automatically. As a result, if I want to move the label, the arrow is moved with it automatically.

All this is possible by using a PERL interface that converts the short command file into an MFPIC source file.

2 The Perl interface

I wrote the PERL script MKPIC (section 5) on-the-fly: I first wrote lines 1–19 and 131–146, which just open, write and close files, define some handy variables and LaTeX commands, and print (line 130) anything in the input file literally to the MFPIC output file. At that point, therefore, commands on the input file had to be valid MFPIC commands. The initial lines also comprised a system call (line 134) running LaTeX, mf and xdvi, so that running the script would display the result. Instead of using a separate input file for my newly created commands, I put them in the __DATA__ section of my script and read them from there. So I had to edit only one file for the creation of both my pictures and new commands.

Then, thinking about how I wanted my pictures to look, I inserted commands in lines 20–128 whenever I felt the need to define one. The first was the *begin* command, of course, which has also the most complex definition, as it defines many scale-dependent variables and TEX commands that might be useful for any command defined later.

Since I defined only what I needed, this PERL script does not have commands for every available MFPIC command. But it is now easy to add more commands.

2.1 How to use mkpic

First of all, read the manpage of the PERL-script, generated from the script using pod2latex, which is shown in section 4.

The easiest way to use the script is to append your own commands to the __DATA__-section of the script, and run it. This will produce a file mkpic.sty, which provides LaTeX-commands named `\Fig<name>`, where `<name>` stands for every name you use in the *begin* command. Finally, you can use those `\Fig<name>` commands in a LaTeX document.

3 Some more examples

Here are a few more examples illustrating some features of the MKPIC script:

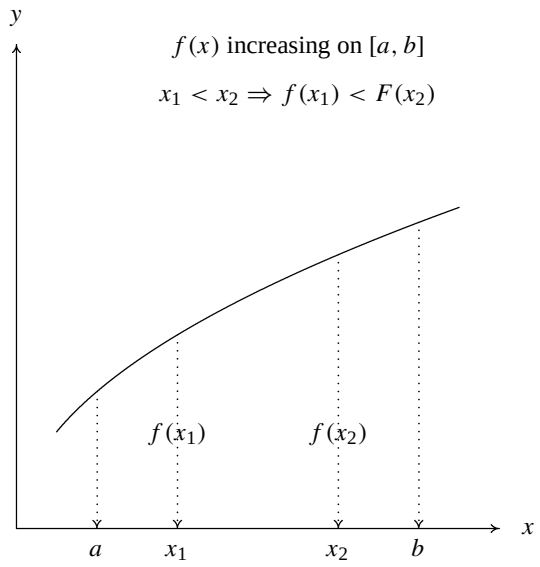


Figure 2. example b

The following commands will produce figure 2:

```
1  begin b 64 64 0 3 12 8 $x$ $y$
   xmark $a$ 2 $x_1$ 4 $x_2$ 8 $b$ 10
   ydrop 2 4.414
   ydrop 4 5
5  ydrop 8 5.828
   ydrop 10 6.162
   label cc 4 4 $f(x_1)$
   label cc 8 4 $f(x_2)$
   label cc 7 8 $f(x)$ increases on $[a,b]$
10 label cc 7 7.5 $x_1<x_2\Rightarrow f(x_1)<F(x_2)$
   func 1 11 .1 x**(.5)+3
   end
```

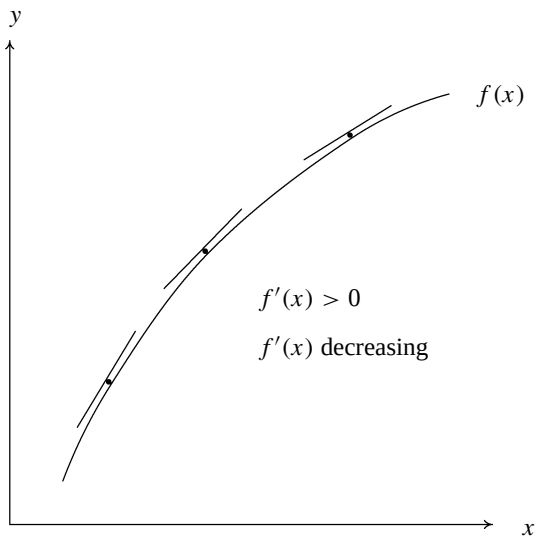


Figure 3. example c

These commands illustrate how valid MFPIC commands can be interspersed between MKPIC commands (see figure 3:

```
begin c 64 64 0 0 10 10 $x$ $y$
curve 1 1 2 3 4 5.7 7 8.1 9 9
\shift{(-.05,.05)}
point 2 3 4 5.7 7 8.1
\shift{(-.05,.05)}
func 1.4 2.6 .1 1.65*x-.3
func 3.2 4.8 .1 1.025*x+1.6
func 6.1 7.9 .1 .62*x+3.76
label cl 9.5 9 $f(x)$
label tl 5 5 $f^\prime(x)>0$
label tl 5 4 $f^\prime(x)$ decreasing
end
```

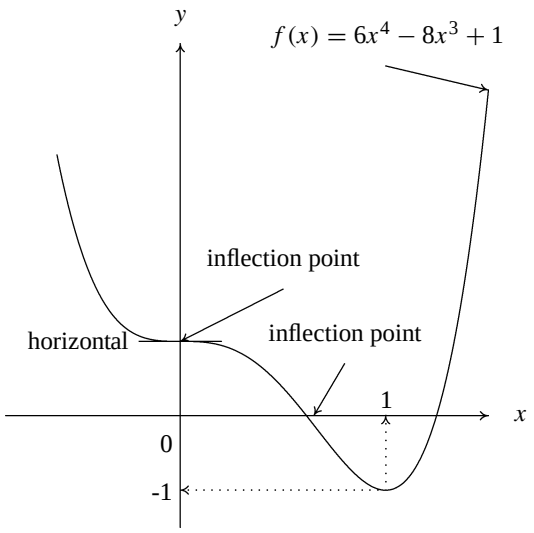


Figure 4. example d

Figure 4 is produced by:

```
begin d 64 64 -.85 -1.5 1.5 5 $x$ $y$
func -.6 1.5 .05 6*(x**4)-8*(x**3)+1
lines -.2 1 .2 1
label cr -.25 1 horizontal
arrow .5 2 0 1 inflection point
arrow .8 1 .65 0 inflection point
arrow 1 5 1.5 4.375 $f(x)=6x^4-8x^3+1$
Xmark 1
ymark \raisebox{-3.5mm}{0} 0 -1
xydrop 1 -1
end
```

And here is an elaborate quasi 3D picture. It shows how comments can be inserted. Standard axes are suppressed because they need special treatment (see figure 5):

```
begin e 64 64 -4 -4 4 4 - -
\dashed
```

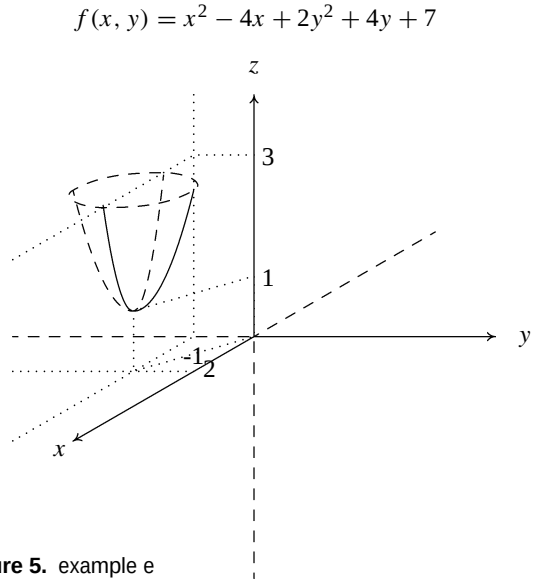


Figure 5. example e

```
lines -4 0 0 0 0 0 -4 # neg z and neg y
\dashed
lines 0 0 3 1.73        # neg x
\arrow
lines 0 0 0 4           # pos y
\arrow[5]
lines 0 0 4 0           # pos z
\arrow[5]
lines 0 0 -3 -1.73      # pos x
\dotted
lines -1 4 -1 0 -4 -1.73  # intersections y=-1 plane
\dotted
lines -1 -.577 -4 -.577
   # intersection x=2 plane with xy-plane
% extra helplines
\dotted
lines -2 .423 -2 -.577 0 0 0 1 -2 .423
Ymark 3
% end of extras
\dotted
lines 0 3 -1 3 -4 1.27
\dashed\sclosed
curve -3 2.42 -1.5 2.711 -1 2.42 -2.5 2.134
label bc 0 \yhi $z$
label cl \xhi 0 $y$
label tr -3.1 -1.8 $x$
label cl -.85 -.577 2
label tc 0 5.5 $f(x,y)=x^2-4x+2y^2+4y+7$
xmark -1
Ymark 1
\shift{(-2,.42)}
\dashed
func 0 .5 .1 9*x*x
func -.5 0 .1 7*x*x
\dashed
func -1 0 .1 2*x*x
func 0 1 .1 2*x*x
end
```

4 The mkpic manpage

NAME

mkpic — interface for making pictures with mfpic

SYNOPSIS

```
begin   name xl yl xmin ymin xmax ymax xlabel ylabel
xmark   [label1] x1 [label2] x2 ...
Xmark   [label1] x1 [label2] x2 ...
ymark   [label1] y1 [label2] y2 ...
Ymark   [label1] y1 [label2] y2 ...
xdrop   x y
ydrop   x y
xydrop  x y
arrow   x1 y1 x2 y2 label
label   YX x y label
point   x1 y1 x2 y2 ...
lines   x1 y1 x2 y2 ...
curve   x1 y1 x2 y2 ...
rect    x1 y1 x2 y2
crect   x1 y1 x2 y2
func    xmin xmax step expression-in-x
#       comment
hatch
bhat
ehat
end
stop
```

DESCRIPTION

mkpic provides an easy interface for generating commands for making small pictures with mfpic. To this end an input file has to be created consisting of commands with space separated parameters.

Currently the following commands are implemented:

begin end Every picture begins with the **begin** command and ends with the **end** command. The **begin** command defines a name for the picture and defines a latex `\newcommand` with that name, prefixed with `Fig`. The resulting `\newcommand` is written to a .sty file. Thus the command

```
begin aa ...
```

starts writing `\newcommand{\Figaa}{...` to the .sty file, and the picture can be reproduced in a LaTeX document by importing the .sty file and using the `\Figaa` command.
xl and yl are the lengths of the x- and y-axes. xlabel and ylabel are the label that are placed at the ends of those axes. Use a space to suppress labeling, or "-" to suppress drawing the axes at all.

xmark ymark Xmark Ymark
These commands place one or more labels along the x- or y-axes, either below (**xmark** and **ymark**) of above (**Xmark** and **Ymark**) the axis.
For the **[xXyY]mark** commands a parameter containing any character other than [−.0-9] is interpreted as the label to be placed and its position is expected in the next parameter. If a parameter is just a number, it is placed at that x-position.

xdrop ydrop xydrop These commands draw dotted arrows perpendicularly to the x-axis, the y-axis and both axes, respectively, ending on the axes with the arrow head.

arrow draws an arrow from *(x1,y1)* to *(x2,y2)* labeled on its tail with *label*

label draws a label at *(x,y)*. *YX* tells how it will be adjusted: for Y=t,b,c *(x,y)* will be, in the y-direction, on top, bottom or center of the label respectively, for X=l,r,c it will be, in the x-direction, left, right or center adjusted on *(x,y)*. Thus

```
label tl 0 0 Hello
```

will draw the string Hello with its lower left corner at (0,0)

point draws points (dots) at (x1,y1), (x2,y2) etcetera.

lines draws line segsments from (x1,y1) to (x2,y2), *(x3,y3)* etcetera.

curve draws a bezier curve from (x1,y1) to (x2,y2), (x3,y3) etcetera.

rect draws a rectangle with diagonal points at (x1,y1) and (x2,y2).

crect clears a rectangle with diagonal points at (x1,y1) and (x2,y2).

func draws the function given by *expression-in-x* between *xmin* and *xmax*, stepping with *step* units in the x-direction.

hatch hatch the closed curve that foloows.

bhat starts a path that will eventually be closed, and then hatched.

ehat ends a path started with **bhat**, closes it and then hatches it.

stop stops further reading of the input. Useful if you have many pictures, but want to see only the first few for testing purposes.

# denotes a comment. The # character and everything following it is discarded.

anything else will be inserted as is in the style file, and therefore should be a valid *mfpic* statement. You use this when you need such a staement only once, or a few times and therefore see no need to define a proper command for it.

5 The Perl script

This is the PERL-script without the pod-text. I removed it as the manpage is shown in a separate section:

```
#!/usr/bin/perl -w

# mkpic - interface for making pictures with mfpic

use vars qw($com);
$tex=shift or $tex='mkpic';

open_stylefile();
for (glob('pictures.*')) {unlink $_}

open(TEX,">$tex.tex");
print TEX '\documentclass[a4paper]{report}
\usepackage{'.$tex.'}
\begin{document}\noindent
';

%pos=('x'=>'tc', 'y'=>'cr', # positions for [xyXY]marks
      'X'=>'bc', 'Y'=>'cl');

while (<DATA>) {
  chomp;
  s/\s*#.*//;           # remove comment
  next unless $_;       # skip empty lines
  /^begin/ and do {
    ($com,$name,$xl,$yl,$xmin,$ymin,
     $xmax,$ymax,$xlabel,$ylabel)=split;
    $xlabel="" if $xlabel eq '-';
    $ylabel="" if $ylabel eq '-';
    $xscale=int(100*$xl/($xmax-$xmin)+.5);
    $yscale=int(100*$yl/($ymax-$ymin)+.5);
    $dx=sprintf("%.2f",100/$xscale);
    $dy=sprintf("%.2f",100/$yscale);
    $yx=$ymin>0 ? $ymin : 0; # y-position of the x-axis
    $xy=$xmin>0 ? $xmin : 0; # x-position of the y-axis
    $xlo=$xy-$dx;     # x-pos of right side of y-markers
    $ylo=$yx-$dy;     # y-pos of top side of x-markers
    $xhi=$xmax+3*$dx; # x-pos of left side of x-label
    $yhi=$ymax+3*$dy; # y-pos of bottom side of y-label
    print
      "%\n%====$name====\n".
      "\\newcommand{\\Fig$name}{\\mftitle{$name}\n".
      "\\def\\xlo{$xlo}\\def\\xhi{$xhi}\n".
      "\\def\\ylo{$ylo}\\def\\yhi{$yhi}\n".
      "\\def\\xy{$xy}\\def\\yx{$yx}\n".
      "\\vspace*{5ex}\n".
      "\\begin{mfpic}[$xscale][$yscale]".
      "{$xmin}{$xmax}{$ymin}{$ymax}\n".
      "\\hatchwd{2}\n".
      "\\tlabel[bc]($xy,$yhi){$ylabel}\n".
      "\\tlabel[cl]($xhi,$yx){$xlabel}\n";
    print TEX "\\mbox{\\Fig$name}\\\\[20mm]\n";
  },next;
  /^arrow/ and do {
    ($com,$x1,$y1,$x2,$y2,$label)=split(/\s+/,$_,6);
    print "\\tlabel[bc]($x1,$y1)".
      "{\\parbox[b]{60mm}{\\center $label}}\n".
      "\\arrow\\lines{($x1,".
      ($y1-$dy*3).".),($x2,$y2)}\n";
  },next;
  /^[xyXY]mark/ and do {
    ($_,@z)=split;
    s/mark//;   # 'x', 'y', 'X' or 'Y'
    for ($i=0;$i<@z;$i++) {
      ($label=$z[$i]) =~ /^[-.\d]+$/ or $i++;
      $x=/x/i ? $z[$i] : /y/ ? $xlo : -$xlo;
      #   ^xmark?        ymark?       Ymark!
      $y=/y/i ? $z[$i] : /x/ ? $ylo : -$ylo;
      #   ^ymark?        xmark?       Xmark!
      print "\\tlabel[$pos{$_}]($x,$y)".
            "{\\strut $label}\n";
    }
  },next;
  /^(point|lines|curve|rect|crect)/ and do {
                  # for example: crect 5 20 20 5
    s/^/\\/;      # \crect 5 20 20 5
    s/\s+/{(/;    # \crect{(5 20 20 5
    while(/\s+/) {
      s/\s+([-.\d]+) ?/,$1),(/;
    }             # \crect{(5,20),(20,5)
    s/..$/}/;     # \crect{(5,20),(20,5)}
    s/crect/gclear\\rect/;
                  # \gclear\rect{(5,20),(20,5)}
    print "$_\n";
  },next;
  /^func/ and do {
    ($com,$x,$y,$d,$f)=split;
    print "\\function{$x,$y,$d}{$f}\n";
  },next;
  /^hatch/ and do {
    print '\rhatch\draw\lclosed';
  },next;
  /^bhat/ and do {
    print '\rhatch\lclosed\connect',"\n";
  },next;
  /^ehat/ and do {
    print '\endconnect',"\n";
  },next;
  /^xdrop/ and do {
    ($com,$x,$y)=split;
    print "\\dotted\\arrow".
          "\\lines{($x,$y),($xy,$y)}\n";
  },next;
  /^ydrop/ and do {
    ($com,$x,$y)=split;
    print "\\dotted\\arrow".
          "\\lines{($x,$y),($x,$yx)}\n";
  },next;
  /^xydrop/ and do {
    ($com,$x,$y)=split;
    print "\\dotted\\arrow".
          "\\lines{($x,$y),($xy,$y)}\n";
    print "\\dotted\\arrow".
          "\\lines{($x,$y),($x,$yx)}\n";
  },next;
  /^label/ and do {
    ($com,$m,$x,$y,$label)=split(/\s+/,$_,5);
    $m=~/^[bct][lcr]$/ or die "illegal label in $_\n";
    print "\\tlabel[$m]($x,$y){$label}\n";
  },next;
  /^end/ and do {
    # axes drawn last for easier rect clears:
    print "\\arrow[\\axisheadlen]\\lines".
          "{($xmin,$yx),($xmax,$yx)}\n" if $xlabel;
    print "\\arrow[\\axisheadlen]\\lines".
```

```
          "{($xy,$ymin),($xy,$ymax)}\n" if $ylabel;
    print "\\end{mfpic}}\n";
  },next;
  /^stop/ and do { last };      # stop reading the input
  print "$_\n" if $_; # anything alse printed literally
}
print TEX "\\end{document}\n";
close(TEX);
close(STY);
system("latex $tex && \
        mf '\\mode=localmode;' input pictures && \
        latex $tex && \
        xdvi $tex");
sub open_stylefile {
open(STY,">$tex.sty"); select STY;
print '\RequirePackage{ifthen}
\input mfpic
\AtBeginDocument{\opengraphsfile{pictures}}
\AtEndDocument{\closegraphsfile}
\setlength{\mfpicunit}{.01mm}
';
}
__DATA__
begin a 64 64 0 -6 4 5 $x$ $y$
xmark 2
Xmark 3
ymark -5 0 4
bhat
lines 0 0 0 4
func 0 3 .05 4-x*x
lines 3 -5 3 0
ehat
func 0 3.2 .05 4-x*x
xydrop 3 -5
arrow 3 4 1.7 1.1 $f(x)=4-x^2$
arrow 2 5 1 2 Area $O_1$
arrow 4 2 2.8 -2 Area $O_2$
end

begin b 64 64 0 3 12 8 $x$ $y$
xmark $a$ 2 $x_1$ 4 $x_2$ 8 $b$ 10
ydrop 2 4.414
ydrop 4 5
ydrop 8 5.828
ydrop 10 6.162
label cc 4 4 $f(x_1)$
label cc 8 4 $f(x_2)$
label cc 7 8 $f(x)$ increasing on $[a,b]$
label cc 7 7.5 $x_1<x_2\Rightarrow f(x_1)<F(x_2)$
func 1 11 .1 x**(.5)+3
end

begin c 64 64 0 0 10 10 $x$ $y$
curve 1 1 2 3 4 5.7 7 8.1 9 9
\shift{(-.05,.05)}
point 2 3 4 5.7 7 8.1
\shift{(-.05,.05)}
func 1.4 2.6 .1 1.65*x-.3
func 3.2 4.8 .1 1.025*x+1.6
func 6.1 7.9 .1 .62*x+3.76
label cl 9.5 9 $f(x)$
label tl 5 5 $f^\prime(x)>0$
label tl 5 4 $f^\prime(x)$ decreasing
end

begin d 64 64 -.85 -1.5 1.5 5 $x$ $y$
func -.6 1.5 .05 6*(x**4)-8*(x**3)+1
lines -.2 1 .2 1
label cr -.25 1 horizontal
arrow .5 2 0 1 inflection point
arrow .8 1 .65 0 inflection point
arrow 1 5 1.5 4.375 $f(x)=6x^4-8x^3+1$
Xmark 1
ymark \raisebox{-3.5mm}{0} 0 -1
xydrop 1 -1
end

begin e 64 64 -4 -4 4 4 - -
\dashed
lines -4 0 0 0 0 0 0 -4  # neg z and neg y
\dashed
lines 0 0 3 1.73         # neg x
\arrow
lines 0 0 0 4            # pos z
\arrow
lines 0 0 4 0            # pos y
\arrow
lines 0 0 -3 -1.73       # pos x
\dotted
lines -1 4 -1 0 -4 -1.73 # intersections y=-1 plane
\dotted
lines -1 -.577 -4 -.577  # intersection x=2 plane
                         # with xy-plane
% extra helplines
\dotted
lines -2 .423 -2 -.577 0 0 0 1 -2 .423
Ymark 3
% end of extras
\dotted
lines 0 3 -1 3 -4 1.27
\dashed\sclosed
curve -3 2.42 -1.5 2.711 -1 2.42 -2.5 2.134
label bc 0 \yhi $z$
label cl \xhi 0 $y$
label tr -3.1 -1.8 $x$
label cl -.85 -.577 2
label tc 0 5.5 $f(x,y)=x^2-4x+2y^2+4y+7$
xmark -1
Ymark 1
\shift{(-2,.42)}
\dashed
func 0 .5 .1 9*x*x
func -.5 0 .1 7*x*x
\dashed
func -1 0 .1 2*x*x
func 0 1 .1 2*x*x
end
```

y2k

Is TEX Y2K-compliant?

Erin Frambach
Rijksuniversiteit Groningen
email: E.H.M.Frambach@eco.rug.nl

abstract

Will TEX and related programs continue to work properly after the year 2000? This article describes how TEX deals with dates and how this affects its behavior after the year 2000.

keywords
millennium, Y2K

What's the problem?

Many programs that have been running perfectly well for a long time could crash or otherwise cause problems at the turn of the century. The reason is that these programs internally represent years in only two digits. This means that, e.g., the year 1999 is represented as '99' and the year 2000 is represented as '00'.

Any program that attempts to do arithmetics with these truncated values will fail. My age, e.g., could be determined by subtracting my year of birth from the current year. In 1999 that would be '99' minus '60' equals '39'. But in the year 2000 it would be '00' minus '60' equals '−60'. Depending on the kind of application that makes such errors, the consequences could be devastating.

If a program behaves decently, using 'large' numbers such as '1999' for internal representations, it is called Y2K-compliant (Y2K means 'Year 2 Thousand'). Many companies are currently testing all their mission-critical software and hardware for Y2K-compliancy. Naturally TEX is also put to the test.

The testing procedure

There are several ways to test if a program is Y2K-compliant. The best way to test software is to scan the source code carefully for any potentially dangerous statements. Secondly, practical tests can be done to verify if a program really behaves correctly when the year turns.

In the case of TEX both tests are needed. The sources of TEX have been publically available for about 20 years and they are exceptionally well documented. This makes it easy to find any potentially dangerous program statements. However, the proof of the pudding is in the eating, so let's set up some tests to see how TEX behaves before, 'during' and after the turn of the century.

Reviewing the sources

TEX's sources are set up as so-called WEB files: ASCII files that contain both the source code and the documentation. The source code can be extracted from the WEB file by a program called Tangle. This general code can be used on any operating system. A 'change file' is provided for operating system specific details, such as a call to obtain the current date. These sources can be compiled into an executable.

In the file TEX.WEB (version 3.14159) we can find the following lines that contain source code dealing with the year value:

```
4922:  @d year_code=23
         {current year of our Lord}
4985:  @d year==int_par(year_code)
5049:  year_code:print_esc("year");
5137:  primitive("year",assign_int,
         int_base+year_code);@/
5138:  @!@:year_}{\.{\\year} primitive@>
10329: print_char(" "); print_int(year);
       print_char(" ");
12262: print(" TeX output "); print_int(year);
       print_char(".");
24125: print_int(year mod 100);
       print_char(".");
```

The TEX primitive command `\year` is defined as a 4-byte signed integer with a range of −2147483648 to +2147483647, which should be quite sufficient. The modulo function applied in line 24125 turns out not to be harmful. It is only used to write non-vital information about the preloaded format to the log file. When running the emTEX 4b implementation a typical log file will show, e.g.:

```
(preloaded format=texput 97.9.3)
```

The (newer) Web2c 7.2 implementation avoids any possible confusion by supplying the full year:

```
(format=tex 1998.3.26)
```

Assuming that the operating system parses the correct year value to the compiler, any integer operation that involves the year value should work correctly.

Practical experiments

In order to prove that TEX outputs the year value correctly we can set up a few tests. The program below uses TEX's variable `\year` to display the current year and a little more information. This program can be run on a computer on which you change the year a number of times. This way you can determine how the program would behave in, e.g. 1990, 1998, 1999, 2000, 2001 and 3000. In any case it should return the same year value as the operating system of the computer dictates.

```
\newcount\YtwoK
\YtwoK = 2000
\advance \YtwoK by -\year
\def\countdown
  {It's \the\year:   % display current year
   \ifnum \YtwoK > 0 % before 2000:
     \ifcase \YtwoK  % 1999
       Time to panic!
     \or             % 1998
       Relax, still so much time.
     \or             % 1997
       Millennium?
     \else           % before 1997
       Don't worry, be happy.
     \fi
   \else             % 2000 or later:
     \ifcase -\YtwoK % 2000
       You've survived!
     \or             % 2001
       Business as usual.
     \else           % after 2001
       When's the next deadline?
     \fi
   \fi}
\countdown
```

Next we can run a program just before midnight 1999 to see what happens if the year turns during the run of a TEX program. The program below will display the current time (in minutes since last midnight), day, month and year in an endless loop.

```
\loop \message{It's now
               \the\time,
               \the\day,
               \the\month,
               \the\year}
\iftrue \repeat
```

From the sources it could be determined that TEX polls the operating system only once to find the current time and date. In a programmer's words: these are not functions but variables that are initialized at the start of the program. Therefore, the test program should write the same line over and over.

We have run these tests on the following implementations of TEX 3.14159:

- emTEX 386 [4b] (MS-DOS, OS/2)
- Web2c 6.1 (Unix)
- Web2c 7.0 (Unix)
- Web2c 7.2 (Windows 95/98/NT)
- Web2c 7.2 (Unix)
- Visual TEX 6.23 (Windows 95/98/NT)
- MiKTEX 1.11 (Windows 95/98/NT)

The test results were identical in all cases.

Related programs

A typical TEX implementation contains many other programs that perform specific tasks. Almost all of them do not perform any task that could induce errors because of an erroneous date value. In fact, dates are hardly used at all in these programs. Nevertheless, a few remarks about these programs are listed below.

Metafont This program supports a date primitive, much like TEX. However, the implementation is slightly different. In MF.WEB we can read that the date value is represented as a 2-byte signed integer. This means that its value can range from −32768 to +32767, which is still quite sufficient.

MetaPost This program is identical to Metafont with respect to date values.

BibTEX This program may be involved in sorting by date. Internally BibTEX stores dates as integers with a more than sufficient range. If a BibTEX user writes a date as, say '98' instead of '1998' then sorting problems may occur. However, it's extremely unusual to refer to an article's year in two digits. In any case, this would a *user*'s problem, not a *program*'s problem.

MakeIndex This program does not use dates at all, so it's unlikely to cause problems.

DVI drivers In general DVI drivers only use dates in output to the console or to a log file, if at all. This should be completely harmless.

Statements

Naturally the Y2K issue has been discussed at the tex-implementors discussion list.

Several TEX implementors have made official Y2K statements about their products. On the Internet these statements are usually easy to find.

Some TEX implementations for which I couldn't find a statement on the Internet I contacted by email. Their responses are also listed below.

tex-implementors discussion list

From: Barbara Beeton
Subject: message for prof. knuth
don,
after a lengthy discussion on the tex-implementors discussion list, it seems advisable for me to write and ask you to consider another change to tex and metafont.

due to the way bureaucracies work, a number of people in the tex community have been asked to sign pieces of paper stating that tex is Y2K compliant.

one of them, chris rowley, had the temerity to point out that there is one place where tex now prints out a year in two-digit format and that perhaps changing this would be very simple and would save a lot of people the time otherwise needed to explain this phenomenon in the current climate of anxiety about such matters.

the code in question is

```
print_int(year mod 100)
```

it is used only in a message giving the date of a format file. chris [Rowley, EF] has further pointed out that in the years 2000–2009, only a single digit will be printed out, a result that may not be clearly recognizable as a year.

the most compelling technical discussion of the details has (as usual) been given by peter breitenlohner:

So much has been said to this subject, that I can't resist...

1. The only place where `print_int(year mod 100)` *is used in tex.web (and something very similar and functionally completely equivalent in metafont and metapost) is when producing the "*`format_ident`*" string. This string is created by initex and written to the .fmt files and displayed (to the terminal and/or transscript file aka logfile). When tex loads a format that string is retrieved from the .fmt file and displayed again. The only purpose of this "*`format_ident`*" string is to somehow identify the format, not nearly in a unique fashion because all formats created on the same day will have the same* `format_ident` *string. Knuth has chosen to use* `(year mod 100)` *as part of that string – probably for human readability – but some sort of hash code would serve the same purpose and might provide a unique identification.*

No one in his clear mind – except possibly a burocrat – should take this use of `(year mod 100)` *as an indication of lacking Y2K compliance!! It would, however, be nicer to use the full year. PT has, however, argued that this is not allowed, since only Don Knuth is allowed to make such changes. I think this point is not quite clear, although* ***it would certainly be best if Knuth either makes or at least approves such a change****. (emphasis added by Barbara Beeton) Otherwise such a modification must certainly live in all the system dependent change files.*

Let me quote from tripman.tex:

> *If somebody claims to have a correct implementation of TEX, I will not believe it until I see that* `\.{TRIP.TEX}` *is translated properly. I propose, in fact, that a program must meet two criteria before it can justifiably be called TEX: (1) The person who wrote it must be happy with the way it works at his or her installation; and (2) the program must produce the correct results from* `\.{TRIP.TEX}`*.*
>
> ...
>
> `\item{5.}` *Compare the* `\.{TRIP.LOG}` *file from step 4 with the "master"* `\.{TRIP.LOG}` *file of step 0. (Let's hope you put that master file in a safe place so that it wouldn't be clobbered.) There should be perfect agreement between these files except in the following respects:*
>
> `\itemitem{a)}` *The dates and possibly the file names will naturally be different.*

The last sentence could be interpreted in the sense that a four digit year in the `format_ident` *string is legitimate.*

2. It has been said that grepping {tex,mf,mp}.tex for "year" is not enough. This is, in principle, certainly correct. Since I know, however, these files fairly well, let me assure you that grepping is good enough in this particular case.

3. tex.web initializes date (`\year`*,* `\month`*, and* `\day`*) and* `\time` *to the 4th July 1776, 12.00 noon. Since tex.web is Pascal based and std. Pascal does not provide such services this is certainly one way to do it. It is then left to the respective implementation (via the system dependent change file) to retrieve the correct date from the operating system and initialize* `\year`*, etc. accordingly.* `\year` *is supposed to be initialized to the current year, e.g., 1998, no truncation!!!*

But: `\year` *et al. are just some of TEX's integer parameters initialized such that they reflect the*

current date and time. They are otherwise completely under the user's control. Using the sequence "`\year=1940 \dump`*" today yields "*`40.10.20`*" as part of* `format_ident`*, "*`\year=2000 \dump`*" yields "*`0.10.10`*", and "*`\year=-143 \dump`*" yields either "*`-43.10.20`*" or "*`57.10.20`*" – depending on how the modulo operation works for negative values (this is not specified in the Pascal Standard).*

with best regards, Peter Breitenlohner

*if a change to remove this pseudo-problem is delayed until 2002, then tex is liable to have to be subjected to a lot of unnecessary scrutiny, taking attention away from things that really *are* problems, and even potentially causing some sites to ban its use. i don't think that is desirable – even less desirable than my writing to you at this date about the problem.*

i hope you agree.

Subject: note from Don Knuth

Dear Barbara,
Thanks for your note. It spurred me to make update my macro files for errata in TAOCP; now they are Y2K compliant!

Peter Breitenlohner's comments are 100% correct. Also, I might note that the Metafont documentation already mentions that Metafont (as it stands) cannot be run after the year 32767; the latter "problem" is not applicable however to TeX.

I agree that it would now be best to remove the "mod 100" from TEX module 1328 and from MF module 1200 (and from MetaPost in the corresponding place). I hereby give permission to implementors to make such changes in their change files. No change to the version numbers are needed.

I've actually made the changes in my personal copies of tex.web and mf.web and tex82.bug and mf85.bug and errata.eight and errata.tex; but those sources won't be updated at labrea until 2002.

I'm not changing page 23 of The TEXbook — where the example was probably my original motivation for cutting to two digits, since the example wouldn't fit on a single line otherwise — nor the corresponding page of The Metafontbook. The format-identifier details are not an essential part of TEX's actions.

Incidentally, I'm not considering this to be the "final bug" in TEX. But it may well turn out to be the final change ever made.

Cordially, Don

On the tex-implementors discussion list Nelson Beebe raised the issue of macro packages that may use truncated year values:

Since `\year` *is accessible to TEX packages, date truncation could also appear elsewhere in TEX distributions. I therefore checked all of the files in the TEX Live 3 tex/tex/latex tree for references to* `\year`*, and turned up just one another year truncation, in:*

`tex/latex/dinbrief/dinbrief.cls`

In this LATEX class file the macro `\ntoday` displays the current date as "day.month.year" in which the year is represented as two digits. Since no calculations are applied on this value, it should be harmless.

PCTEX

Their official statement can be found on `http://www.pctex.com/techsupp/itechinf.htm` and this is what it says:

"Is PCTEX Year 2000 Compatible?

Because PCTEX and accompanying macros etc. are operating system/date dependant; as long as you have taken proper precautions to ensure your operating system is year 2000 compliant, you will experience no change in operation, come January 1, 2000. If you need a verification letter for your files, click HERE"

The word *HERE* above links to `ftp://ftp.crl.com/users/rw/98765/yr2000.dvi` which reads:

"June 16, 1998

Year 2000 Manager

To Whom it May Concern,

The vendor warrants that each item of hardware, software, and/or firmware delivered, developed or modified under this contract shall be able to accurately process date data (including, but not limited to calculating, comparing and sequencing) from, into, and between the twentieth and twenty-first centuries, including leap year calculations, when used in accordance with the item documentation provided by the contractor, provided that all items (e.g. hardware, software, firmware) used in combination with other designated items properly exchange date data with it. The duration of this warranty and the remedies available to the customer for breach of this warranty shall be as

defined in, and subject to, the terms and limitation of any general warranty provision(s) of this contract, provided that notwithstanding any provision to the contrary in such warranty provision(s), or in the absence of any such warranty provision(s), the remedies available to the customer under this warranty shall include repair or replacement of any item whose non-compliance is discovered and made known to the contractor in writing within ninety (90) days after acceptance. Nothing in the warranty shall be construed to limit any rights or remedies the customer may otherwise have under this contract with respect to defects other than Year 2000 performance.

Sincerely,

Lance Carnes
President"

TrueTEX

Their statement can be found on `http://www.truetex.com/y2k.htm` and it reads:

"TEX Year 2000 (Y2K) Issues in Summary

This page summarizes the year 2000 issues surrounding TEX and METAFONT, based on discussions in the newsgroup comp.text.tex and among the members of the mailing list `tex-implementors@ams.org`. We attempt herein to merely set forth the chief matters at hand, without engaging the controversial aspects of what solutions should be taken.

1. *Crashing: The programs TEX and METAFONT themselves will not crash due to dates. (However, each executable implementation depends on a run-time library and an operating system, which should be evaluated in this regard.)*
2. *Timestamps: A 2-digit year is (a) printed in logfiles, and (b) stored in format file and base file time stamps. These items should not be of general concern, because they are intended for human readers and not as input to other programs.*
   On November 24, 1998, Donald Knuth granted an unusual permission to modify TEX and METAFONT to use 4-digit timestamps (nearly all implementations, such as web2c, had already been doing so), saying:

   > *I agree that it would now be best to remove the "mod 100" from TEX module 1328 and from MF module 1200 (and from METAPOST in the corresponding place). I hereby give permission to implementors to make such changes in their change files. No change to the version numbers are needed. [As reported by Barbara Beeton on the tex-implementors e-mail list.]*

   This permission means that 4-digit timestamps, while changing the output of TEX and METAFONT slightly from the current autographs, still meet Knuth's authoritative standards required of software calling itself TEX or METAFONT.
3. *The `\year` primitive: TEX TRIP certification, in the strictest sense, does not require that `\year` return a meaningful value (TEX may be certifiably implemented on platforms that do not even supply date reporting, such as standard Pascal). The TEXbook does define `\year` as "the current year of our Lord", which is the only correct meaning of `\year` for those implementations which can supply a meaningful value, which is to say nearly all of them.*
   In short, TEX implementations should provide a value in `\year` giving the 4-digit year Anno Domini, or the value 1776 if the platform does not support a date function. TEX does not provide any state variables to indicate whether `\year` contains a meaningful value, and while 1776 could have been considered a signal value for a lack of meaning to `\year`, this is not a standardized requirement.
4. *External software: The TEX corpus embodies many accessory programs, such as macro packages and DVI translators, which may compute dates from the value of `\year` (or rarely, from timestamps). Such accessories should be checked individually for correct behavior when `\year` is assumed to return a correct 4-digit `\year` value before and after 2000. Accessories with an additional "defensive level" of correctness will behave reasonably when `\year` contains a two-digit value or a meaningless value such as 1776."*

Y&Y

Their statement can be found on `http://www.YandY.com/Y2K.htm` which reads:

*"**Year 2000 (Y2K)***

Relative to the Year 2000, there are no date issues with Y&Y TEX System release 2.1

Y&Y TEX uses only standard calls to the Microsoft C++ library and WIN32 API to manage the dates and times for files. The operating system sets the date and time arithmetic values based on what the PC's realtime clock provides. It is the operating system, the underlying BIOS, and the realtime clock upon which Y&Y software depends for Year 2000 compliance (*)

We do not embed or implement any algorithm of our own or any third party's to compute dates in any of our products.

TEX itself does not do any computation based on dates. It simply provides informational access to the dates and times obtained from the underlying operating system. The year is not truncated to two digits when offered to the user from inside TEX source code via `\year`*, or in information output on screen or in the log file. This information is provided "AS IS" without warranty of any kind, either express or implied. Any further question may be addressed to the attention of* `support@YandY.com`

(*) Some C libraries on Unix, DOS, Windows and Macintosh will have an overflow problem in the year 2038 or 2040. Unlike some other TeX systems, some browsers, and some email programs, the Y&Y TEX System release 2.1 will not crash when run on a machine with the year set to these values."

In an email Bertold Horn, representing Y&Y, wrote to me:

"*We some time ago switched to 4 digit years in the comment string in the format file. Which is the only place where there is anything even remotely related in TEX itself. We did not feel any need for wait for Knuth to make a statement on this. In fact we have made numerous such small changes to improve TEX.*

[...]

IMHO, the Y2K problem is vastly overrated. The year 2038 and 2040 bugs are more serious. Unfortunately I won't be around to have to worry about that.

Regards, Berthold Horn."

Visual TeX

Through email MicroPress Inc. stated:

"*No statement, besides "VTEX is fully Y2K compliant"*

A statement of Y2K compliance will appear on the Web site in the near future. (With all the new things, the site has fallen behind a bit.)"

Unfortunately the full statement is still not available.

BlueSky

BlueSky's official statement can be found on `http://www.bluesky.com/y2knote.html` and it reads:

"*Year 2000 Compliance*

As you may or may not know, the Macintosh in general has been Y2K compliant since it was first introduced in 1984. Contrast this with Microsoft's Windows software—even the Windows 95 version is not Y2K compliant, nor is their Windows NT 4.0! See Apple's year 2000 page on the web.

Likewise, most Macintosh software is built to handle the year 2000, and Textures is no exception. Don Knuth, the author of the TEX typesetting system and the multi-volume set "The Art of Computer Programming" was well ahead of the curve, and TEX has been Y2K compliant since the 70's!

In short: Yes, Textures is Y2K compliant.

Apple Macintosh: the choice for the year 2000 and beyond ;-)"

emTEX

In an email Eberhard Mattes wrote the following about the millennium problem:

"*AFAIK, the only problem is TEX printing only two digits of the year in the .log file. (Blame DEK.) Completely harmless.*"

Conclusions

Studying the TEX sources shows that a Y2K-problem is very unlikely to appear. The practical tests revealed no problems either. TEX jobs that do calculations on dates should not be run around midnight because TEX only polls for the current time and date once, at the start of the job. But that is only natural for a program that's not supposed to run continuously: this limitation has no relation to Y2K-compliance.

TEX is a programming language that can be used to program just about anything, including applications that are clearly not Y2K-compliant. TEX itself does not do any calculations that involve date values. The value of the variable `\year` is an integer (with a range more than sufficient for any year value) that is provided by the operating system on which TEX runs. Therefore, we have every confidence that, when TEX is run under a Y2K-compliant operating system, it will itself be compliant. However, being freeware, no warranty (express or implied) accompanies its sources, and TEX users are therefore advised to perform their own examination and testing.

database application

A Database for PPCHTEX

Richard Müller
(r.a.mueller@cityweb.de)

abstract

A database with PPCHTEX code would ease the work for those who have to draw chemical formulas only occasionally and have no time to become acquainted with it. But experts could also make use of it. The establishing of such a database is discussed and proposed.

keywords

PPCHTEX, chemistry, structural formulas, database

Some time ago I came around to draw some chemical formulas. Since I am using TEX for the work which should look nicer than with ordinary textsystems I looked around to find something to set these formulas with. I found out that PPCHTEX would do the job quite well. Then I started to try to understand the language you use for PPCHTEX. This was not as simple as it first seemed. Why does, for example, that sidegroup not appear on the screen? What was wrong with the code? In my profession (I am a biology teacher) I don't use heaps of chemical formulas, but the occasional one or other. So you don't get very acquainted with the PPCHTEX-language. In OTTEN (1998) and CONTEXT (1998) I found help. But both publications have disadvantages: The manual (OTTEN 1998) is essential to understand and write proper PPCHTEX-code. But complete examples are rare and can be used unchanged only in very seldom cases. And of course this is not the intention of the article; it's a manual and not a dictionary of formulas. CONTEXT up-to-date 1998/2 is a fine PDF-document with estimated 170 examples of nice-looking formulas. But unfortunately no names are given, so you can find the structures you are searching for only by clicking from one page to the next through the whole document. In case names were given you could use Acrobat's search-function. And as far as I know it is not intended that further examples become added to this collection, be it by the authors or be it by the users. One day as I was figuring out a new structure I thought "Are you the only one who uses this wicked language?" The idea came on to me, that all users of PPCHTEX should join their codes and establish one pool which is used and also being contributed to by everyone who uses PPCHTEX. It is obvious that the right place to publish such a code collection is the internet. To faciliate search and navigation you would need a hypertext system as PDF or HTML. The choice fell on HTML because it is integrated in every browser and office software. I simply used StarOffice to set up my example pages quite quickly. (Perhaps I could have used PDFTEX, but I have no experience with it). What should the pages of the database contain? The answer is quite simple: name, formula and structure! To cover synonyms, every known and used name of a chemical substance covered in the database should appear in the alphabetical index, be it perhaps the systematical IUPAC name, be it its common name, or be it a name used in former times. These different names for the same substance should be linked with the same page: *propanetriol* and *glycerol* are linked with the page `glycerine.html`. On the other hand, in every case you can write the same substance in different manners: vertically or horizontally orientated, with carbon atoms in the corners or without, in ring or in chair form and so on. All these different examples of the same substance should be collected on the same page, whith the typography of the formula shown together with its specific code. In a discussion with Hans HAGEN the idea was born to publish only the bodies of the structures. The user should then complete the code of the desired structure himself. In my opinion this is possible with some substances which differ e.g. only in sidegroups: Most types of *dioxines* have nearly the same structure. The sidegroups are represented by letters in alphabetical order which can be found in the published code. So it is simple to replace the letters in the code with the desired groups or atoms. But how to print the exact typography of the formula in the database? I found no other way than to perform a LATEX run on the code and to print the structure on the screen. I used Windvi and Ghostscript with Ghostview. As far as I could distinguish, the formula printed by Ghostscript looked a little bit nicer on the screen. With screen capturing I made a GIF of the formula which became a part of my HTML page. To save space (and loading time) the GIF should be as small as possible. The GIFs I present have filesizes between 2 and 5 kB.

This database cannot live solely with my own contributions. In the beginning it will be quite empty, but with a little help of PPCHTEX' friends it should become filled in a certain amount of time. I want to find out, if there is need for such a project and if other PPCHTEX users are willing to share their codes with the rest of the community. In case

this is true, the following topics should become discussed: Can the proposed form of the database suit the needs of the user? Or ist it perhaps of more advantage to create a kind of mailinglist restricted for publishing PPCHTEX code, where everyone simply can contribute by mailing the name, code and GIF? In case we vote for establishing an internet-based database as pointed out above, should it be open for direct writing into it by everybody? I am sure you find a lot more points worth discussion. You can navigate to my example pages of the proposed database at:
`http://www.caw.de/walram/ppchtex/00index.html`.
I appreciate your comments.

Literature

CONTEXT 1998: CONTEXT up-to-date 1998/2. Pragma ADE, Hasselt 1998. (`http://www.ntg.nl/context/uptodate.htm`)
OTTEN, T. 1998: PPCHTEX manual. MAPS **20**, 150, 1998

Beginnen met ConTEXt

Berend de Boer

Volgens Hans zou ik de eerste ConTEXt gebruiker buiten Pragma zijn. Ik heb dat eens nagekeken: ik heb hier nog een context format uit 1996. Dat zou dan de eerste `context.fmt` zijn die Pragma heeft verlaten, een collectors item :-).

Ik gebruik TEXsinds 1989 of 1990. Ik geloofde toen nog dat WordPerfect 5.1 het eind van alle tekstverwerkers was. Maar laten we dat maar snel vergeten, het helpt me in elk geval begrip op te brengen voor Word gebruikers...

Voor veel documenten maakte ik custom layouts. Dat hield dus in LATEX sources aanpassen. LATEX is slecht aanpasbaar of instelbaar, dus meestal kwam dat neer op het kopiren van veel macro's uit `latex.fmt` en een aantal niveaus vervolgens hacken. Bij de volgende versie van LATEX begon het spel weer opnieuw.

Toen ik Hans ontmoette beweerde hij dat dat allemaal veel eenvoudiger kon. Ik ben op een avond bij Pragma geweest met een layout die mij bloed, zweet en tranen had gekost. Voor een shareware pakket had ik de layout van Borland's handleidingen gemplementeerd. Ik was vrij sceptisch over hoe Hans zou laten zien dat dit in een handomdraai nagemaakt kon worden. Verbazen zal ik de ConTEXt gebruikers niet: Hans kon dat inderdaad heel erg snel.

Op die avond was ik om. Ik kreeg een ConTEXt handleiding, format files en een speciale versie van emTeX: HugeTex. Thuis werkte alles vrijwel meteen. Toch heeft het nog vrij lang geduurd voordat ik vrijwel uitsluitend ConTEXt gebruikte. Ik stapte in die tijd ook over op Unix als mijn primaire platform. ConTEXt 'draaide' nog niet onder Unix omdat Hans de sources nog niet had vrijgegeven, alleen de `.fmt` bestanden. En het valt niet mee als je erg gewend bent aan bepaalde macro's om dan opeens alles te doen met geheel andere macro's. Ik was dan ook in het begin veel tijd kwijt met het opzoeken van hoe stel ik dit in en hoe stel ik dat in.

Ik denk dat dit laatste dan ook de grootste hobbel is die ConTEXt zal moeten nemen. Te veel mensen hebben met LATEX leren leven. Met dit artikel probeer ik daar een beetje verandering in te brengen door te laten zien hoe je met ConTEXt layouts kunt instellen.

De volgende zaken komen aanbod:

1. Hoe stel je ConTEXt standaard in dat je PDF uitvoer krijgt.
2. Hoe wijzig je het papierformaat.
3. Hoe stel je een basis font in.
4. Hoe wijzig je de hoofdstuk en paragraaf kopjes.
5. Waar komt het paginanummer.
6. Hoe krijg ik eigen hoofd– en voetteksten.
7. Kan ik ook nog ergens mijn tekst intypen asjeblieft?
8. Hoe voeg je een plaatje toe.
9. Hoe genereer je een inhoudsopgave.
10. Hoe genereer je een index.

Ik richt me in dit verhaal vooral op gebruikers die PDF uitvoer willen.

Verder ga ik uit van de allerlaatste ConTEXt versie. Voorgaande versies deden dingen anders.

PDF uitvoer

De eerste beslissing die ik altijd neem is PDF output te gebruiken. De documenten die ik maak zijn vrijwel uitsluitend bestemd voor een publiek dat niet over een dviviewer beschikt. PDF is ideaal als platform onafhankelijke uitvoer.

Als eerste regel in een document neem ik dus dit op:

```
\steluitvoerin[pdftex]
```

Omdat ik dit voor alle documenten wil, heb ik dit commando in een speciaal bestand geplaatst, namelijk `cont-sys.tex`. Dit bestand leest ConTEXt bij het opstarten.

Paginaformaat instellen

Het paginaformaat is het volgende commando. Dat is in te stellen met:

```
\stelpapierformaatin[A4][A4]
```

De eerste parameter legt het formaat vast waarmee ConTEXt werkt bij het typesetten van de tekst. De tweede parameter legt het formaat vast wat uit de printer rolt. Veel meer keus dan A4 is er, praktisch gezien, meestal niet.

Fonts instellen

De volgende stap is het instellen van het basisfont. Dat kan met dit commando:

```
\stelkorpsin[ber,phv,ss,11pt]
```

De eerste parameter geeft aan dat we de Karl Berry encoding hanteren, volgens mij de meest gebruikelijke hantering voor postscript fonts. De aanduiding “ber” is dus zowat verplicht als je je eerste postscript font opgeeft, daarna is deze niet meer nodig.

De tweede parameter geeft aan dat we het font Helvetica willen laden, een font zonder schreefje. Dit font is ongeveer gelijk aan Arial, wat Windows gebruikers wellicht beter kennen.

De derde parameter geeft aan ConTeXt door dat we nu gaan switchen naar dit font–zonder–schreefje, “sans serif” dus. Doordat we eerst Helvetica hebben geladen is het standaard ‘ss’ font overschreven.

De derde parameter geeft de font grootte op.

Een voordeel van een standaard postscriptfont is dat het PDF bestand kleiner blijft. Er hoeft geen extra font informatie in te worden opgenomen.

Ruimte tussen regels instellen

Als het font wat groter is als het standaard font van 10 punten, dan is het nodig om ook de regelafstand wat aan te passen. Het volgende commando zorgt er voor dat de standaard regelafstand iets kleiner wordt.

```
\stelinterliniein[regel=13pt]
```

Ruimte tussen paragrafen instellen

Meestal wil ik een lege regel tussen paragrafen. Dat kan met dit commando:

```
\stelwitruimtein[groot]
```

Instellen van hoofdstuk en paragraaf kopjes

De standaard kopjes zijn niet vet. Met het volgende commando is het mogelijk om de kopjes van de hoofdstukken, paragrafen en subparagrafen vet te zetten.

```
\stelkopin[hoofdstuk][letter=\bfc]
\stelkopin[paragraaf][letter=\bfa]
\stelkopin[subparagraaf][letter=\bf]
```

De eerste parameter verwijst naar hetgene wat je wilt instellen: hoofdstuk, paragraaf, subparagraaf, subsubparagraaf, etc.

De tweede parameter is de instelling zelf. Hier wijzig ik alleen de letter. Maar er zijn nog veel meer instelling. Met het volgende commando plaats je bijvoorbeeld de nummers in de marge:

```
\stelkopin[hoofdstuk][letter=\bfc,variant=inmarge]
\stelkopin[paragraaf][letter=\bfa,variant=inmarge]
```

Het is ook mogelijk om de kop geheel zelf te formateren. Bijvoorbeeld het hoofdstuknummer kan als volgt geformatteerd worden:

```
\def\kophfd#1#2{{\bfc Nummer: #1, Titel: #2}
\stelkopin[hoofdstuk][commando=\kophfd]
```

Instellen van paginanummer

Het paginanummer komt standaard middenin, bovenaan te staan. Het volgende commando plaatst het paginanummer onderaan.

```
\stelnummeringin[plaats={midden,voet}]
```

Het paginanummer wordt met dit commando uitgezet, want in de volgende paragraaf worden hoofd– en voetteksten ingesteld met paginanummer.

```
 \stelnummeringin[plaats=]
```

Als het paginanummer niet uitgezet wordt, komt het over de hoofd– en voetteksten heen.

Instellen van hoofd– en voetteksten

Het is eenvoudig zelf hoofd– en voetteksten in te stellen. De volgende twee commando’s definiren de opmaak. De laatste twee commando’s maken deze opmaak actief.

```
% eigen koptekst
\def\mykoptekst{%
 \vbox{%
  \hbox to \tekstbreedte{\bfb Test\hfill Pagina}
  \par\haarlijn
 }
}
% en eigen voettekst
\def\myvoettekst{%
 \vbox{%
  \haarlijn\par
  \hbox to \tekstbreedte{%
   \huidigedatum
   \hfill
   {\bf Mijn bedrijf}\hfill
   Pagina~\paginanummer~van~\totaalaantalpaginas
  }
 }
}
% instellen
\stelhoofdtekstenin[\mykoptekst]
```

```
\stelvoettekstenin[\myvoettekst]
```

Voor degenen die dit niet kennen: met het commando `\hfill` is het mogelijk om dingen naar links en naar rechts te ‘duwen’. Een `\hfill` wordt net zo groot als mogelijk. Neem je twee `\hfill`’s op, dan worden ze allebei even groot. Dus de opmaak in `mykoptekst` duwt de tekst ‘Test’ naar links en de tekst ‘Pagina’ naar rechts totdat ze allebei tegen de kantlijn aanstaan, en dus niet verder geduwd worden.

Om ze tegen de linker en rechterkantlijn te krijgen wordt een horizontale box gemaakt waarbinnen dit allemaal gebeurd. Die horizontale box is net zo groot als de breedte van de tekst.

Als deze box, commando `\hbox`, niet geplaatst zou zijn, kreeg je niet het gewenste effect, want dan zou de ruimte rechts ‘beperkt’ worden door de `\par`.

De tekst zelf

Na al deze instellingen zou je de tekst zelf bijna vergeten! Tekst wordt geplaatst na het commando `\starttekst`. De tekst eindigt met `\stoptekst`.

```
\starttekst

Hier komt mijn tekst.

\stoptekst
```

Hoofdstukken en paragrafen worden als volgt gedaan:

```
\starttekst

\hoofdstuk{Mijn hoofdstuk}

\paragraaf{Mijn paragraaf}
Hier komt mijn tekst.

\stoptekst
```

Verwijzen naar paragrafen kan met het `\in` commando.

```
\starttekst

\paragraaf{Mijn eerste paragraaf}
Hier komt mijn tekst, zie ook
paragraaf~\in[par:twee].

\paragraaf[par:twee]{Mijn tweede paragraaf}
En hier nog meer.

\stoptekst
```

De paragraaf krijgt dus een extra parameter, tussen rechte haken. Tussen die rechte haken komt een volkomen willekeurige tekst waarnaar verwezen kan worden met het `\in` commando.

Opnemen van een plaatje

Plaatjes opnemen is heel eenvoudig geworden met de laatste ConTEXt en `pdftex` versies. Neem dit commando op om een plaatje te definiren:

```
\gebruikexternfiguur[MijnPlaatje][]
                    [type=png,schaal=600]
```

De eerste parameter is de naam van het commando waarmee je dit plaatje op kunt roepen. De tweede parameter is de naam van het bestand van dit plaatje. Als dit leeg gelaten wordt, is dit gelijk aan de eerste parameter.

Met de derde parameter kan het type opgegeven worden, en kan het plaatje eventueel geschaald worden. Dit is afhankelijk van de TEX versie die gebruikt wordt. Met `pdftex` werkt alles prima.

Het plaatje kan daarna opgeroepen en geplaatst worden met:

```
\plaatsfiguur
 [hier]
 [fig:MijnPlaatje]
 {Een mooi plaatje}
 {\MijnPlaatje}
```

De eerste parameter geeft aan waar het plaatje geplaatst mag worden.

De tweede parameter is een referentie waarmee naar het plaatje verwezen kan worden, zie onder. Een referentie is een volkomen willekeurige tekst.

De derde parameter is de bijbehorende tekst.

De vierde parameter is het commando voor het plaatje zelf.

Grootte e.d. worden automatisch bepaald door ConTEXt mits een recente TEX gebruikt worden.

Verwijzen naar dit plaatje kan via:

```
Figuur~\in[fig:MooiPlaatje] bevat een
mooi plaatje.
```

Genereren van een inhoudsopgave

Om de inhoudsopgave te kunenn maken zijn twee slagen nodig. Eerst moet ConTEXt het bestand een keer verwerken. Er wordt dan een bestand met de extensie `.tui` gecreerd. Dit bestand kan met `texutil` verwerkt worden:

```
texutil --references test
```

Als ik dat intype, krijg ik de volgende output:

```
/home/berend/tmp# texutil.pl --references test
```

```
 TeXUtil 6.6 - ConTeXt / PRAGMA 1992-1998

               action : processing commands, l\
                                ists and registers
           input file : test.tui
          output file : test.tuo
       passed commands : 6
      register entries : 0 -> 0 entries 0 refer\
                                              ences
       synonym entries : 0 -> 0 entries
        embedded files : 1
/usr/home/berend/tmp#
```

`texutil` is een commando dat bij de ConTEXt distributie geleverd worden.

Daarna kan de inhoudsopgave geplaatst worden door dit commando op de gewenste plek neer te zetten:

```
\volledigeinhoud
```

Voor mensen die de layout van de inhoudsopgaven willen instellen is er het `\stelsamengesteldelijstin` commando.

```
\stelsamengesteldelijstin
 [inhoud]
 [variant=c,
        niveau=subparagraaf]
```

Genereren van een index

Een index wordt op dezelfde wijze als de inhoudsopgave gegenereert. Eerst moet ConTEXt het bestand een keer verwerkt hebben, daarna kan `texutil` met parameter `--references` aangeroepen worden.

De inhoudsopgave wordt met het volgende commando geplaatst.

```
\volledigregister[index]
```

Om iets in de index te krijgen, zijn er twee commando's. Met `\index` kan iets in de index geplaatst worden. Met `\zieindex` kan een verwijzing binnen een index gemaakt worden.

```
\zieindex{hallo wereld}{hello world}

\starttekst
De meest bekende test--tekst is
wel \index{hello world}‘hello world’
\stoptekst
```

Het commando `\zieindex` zal meestal in het instelgebied, dus voor `\starttekst` terecht komen. Het commando `\index` wordt gebruikt *voor* de tekst waar deze naar verwijst.

Waar is meer informatie te vinden

Ik heb ConTEXt vrijwel uitsluitend geleerd met behulp van de handleiding in `ms-co-nl.pdf`. Voor de meeste instellingen vond ik deze handleiding voldoende.

Verder gebruik ik veel het tooltje `grep`. Dit tooltje kan woorden opzoeken in tekst. Wil ik bijvoorbeeld weten waar in de ConTEXt sources het commando `\stelpapierformaatin` voorkomt, dan type ik:

```
/usr/local/share/texmf/tex/context/base# grep -n\
                     "stelpapierformaatin" *.tex

core-01a.tex:30:% \stelpapierformaatin[liggend,A\
                                    4][liggend,A4]
core-01a.tex:1649:%I   \stelpapierformaatin[DIN-\
                                        formaat]
core-01a.tex:1661:%I \stelpapierformaatin[A5][A4]
core-01a.tex:1708:\def\dostelpapierformaatin[#1]\
                                          [#2]%
core-01a.tex:1744:  \stelpapierformaatin[#1][#2]%
core-01a.tex:1747:\def\stelpapierformaatin%
core-01a.tex:1748:  {\dodoubleempty\dostelpapier\
                                      formaatin}
...
```

Vervolgens kan ik in 1 van de opgesomde bestanden kijken of er nog iets extra's over dit commando vermeld is, wat nog niet in de handleiding staat. De sources lezen doe ik nauwelijks. TEX sources zijn vrij onleesbaar wat mij betreft, maar je krijgt wel de indruk bij de ConTEXt sources dat ze leesbaar zijn, een heel verschil als je `latex.ltx` bestudeert hebt.

Conclusie

Ik hoop dat dit artikel mensen heeft aangezet om ConTEXt eens uit te proberen. Ik geef toe dat TEX pakketten voor de gemiddelijke gebruiker niet makkelijk te installeren zijn. Maar voor wie de luxe heeft een systeembeheerder te hebben: vraag hem om ConTEXt ook beschikbaar te maken. Dit kost hem waarschijnlijk weinig moeite.

Voor mensen die verder vragen en of opmerkingen hebben over ConTEXt: welkom bij de ConTEXt mailling list. Stuur een mailtje naar `majordomo@ntg.nl` met als inhoud “subscribe ntg-context”.

advanced

Optimizing TEX code

some words on speed and space

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

abstract

Macros can be collected in macro packages. These packages can be stored in a form that permits fast loading. Although TEX is already pretty fast, for demanding applications it makes sense to speed up TEX to the max. Switching to e-TEX and beyond is one way to achieve this, another way can be found in optimizing the macro code by means of a dedicated program. Currently the combination of both can speed up TEX runs by at least 10%.

keywords

speeding up, formats, optimizing code, e-TEX, ConTEXt

Wat is TEX?

TEX is a typographic computer language with a strange character. To mention one: typographical programs and text to be typeset can be mixed. Therefore the border between programs and text is not always that clear. Take

```
1 2 3 \hbox spread 1em{\hss4\hss} 5 6 7
```

which shows up as

1 2 3 4 5 6 7

This piece of code does not really contain a program, but the next example does:

```
\setbox0=\hbox{12}\dimen0=\wd0

1 2 3 \hbox to \dimen0{\hss4\hss} 5 6 7
```

As long as one digit is used, the result is the same as in the previous example, because a digit has a width of .5em. It makes sense to isolate textual input and programming code, like in:

```
\def\TwoDigitsWide#1%
  {\hbox to 1em{\hss#1\hss}}

1 2 3 \TwoDigitsWide{4} 5 6 7
```

The `\def`'d things are called macros. Quite often macros are collected into so called macro packages. That way we save ourselves the time of retyping commonly used macros and at the same time force consistency.

So, TEX is a programming language and a program written in such a language should either be compiled into some low level machine code or interpreted at runtime. Again, TEX is a strange breed, because it does a bit of both: TEX the programming language is processed by TEX the program. While read in from a file, characters and sequences of characters are translated into an internal representation and when not directly typeset, they are stored in appropriate data structures.

In our example, the definition (`\def\Two...`) is stored for later use, and called upon while `1 2 ..` is typeset.

Running TEX

On many computer systems users invoke their prefered macro package by typing in its name at the command line and often they think that `tex`, `latex` or `context` is simply a program. This situation becomes even more vague when one encounters `pdftex` and `pdflatex`. (Be suspicious if you ever encounter `pdfcontext`, because by nature CONTEXT is not aware of any specific brand of TEX, except E-TEX!)

What actually happens is that TEX the program is launched and instantly starts reading in the macro package one asked for. So, it's still TEX that one's running, no matter in what way it is invoked! On many systems saying:

```
tex      <filename>
latex    <filename>
context  <filename>
```

is just the same as:

```
tex &plain   <filename>
tex &latex   <filename>
tex &context <filename>
```

which is the 'official' way of calling TEX with some kind of macro package.

Formats

The macros normally are not stored in ASCII format, but in a so called memory dump, a sort of precompiled format. One can generate such a format by saying:

```
tex --ini plain.tex \dump
tex --ini latex.ltx
tex --ini context.tex
```

When run in ini (also called virgin) mode, TEX reads in the macro package and when finished, it dumps its memory on disk. Such a dump normally has the suffix `fmt` and can be reloaded fast: `&plain` refers to the format. It does not hurt to know this! Try it, if only to be able to generate an updated format.

So, to summarize, we have TEX the programming language, TEX the program, and all kinds of macro packages, that can be packed into memory dumps for fast loading.

When generating an PDFTEX format, or E-TEX or PDF-E-TEX format, a slightly other method applies: PDFTEX has it own configuration information, while E-TEX only has additional functionality when the filename is prefixed by a `*`. To ease the life of CONTEXT users, there is TEXEXEC. This PERL script wraps all brands of TEX in a shell, so there we say:

```
texexec --make en
```

to generate a format with an english user interface. Running CONTEXT is done by saying something like:

```
texexec myfile
texexec --pdf myfile
```

So, when you're using CONTEXT, don't worry if all this formatting talk is beyond you: TEXEXEC is there to guide you. So far for this interlude.

Tokens

TEX thinks in terms of tokens. Let's for this moment forget about the sometimes pretty confusing way TEX reads and interprets the characters, and consider the next example:

```
\expandafter\def\csname name\endcsname{...}
 1          2    3   4 5 6 7    8      9 11 13
```

Once read in memory, this example is represented in 13 tokens. The names of macros as well as the `{`, `}` and .'s each become one internal quantity. The space after `\csname` ends the control sequence and does not count. I will not go into details on the amount of memory each token takes, but in general, one can say that TEX tries to minimize the amount of memory used. A macro name is stored in the so called hash table (a lookup table) and after that only references to this table are used. In the TEX that I'm currently running, a reference to a macro name takes 8 bytes: 4 bytes for the index, a pointer to the next token, some space for an additional index (used by `\charsubdef`) and a rounding byte for efficient internal representation.

The main point of this short story is that once read in, TEX does not need to translate macro names, but simply uses pointers to reach the meaning of this macro. This is not only faster, it also takes less memory. And this is why it makes sense to use memory dumps instead of reading in a macro package each time. TEX loads faster and runs faster.

One should be aware of the fact that we're only talking of a clever way of storing data. Opposite to for instance high level programming languages like Pascal and C, no compilation is done. The meaning of the macro is stored as it is, even if its meaning is wrong in some way or another. Nearly nothing is checked and nothing is optimized. And this is one of the reasons why TEX, while being an interpreter, is so fast.

The main reason why I could give CONTEXT a multi lingual interface without having to recode those tens of thousands lines of code, is that I used a rather high level of abstraction. Keywords and values are stored as macros. This is a sort of lucky coincidence: when CONTEXT grew, TEX's were still pretty small, so I ran out of string memory (the total length of all strings used) before I ran out of hash memory (the number of macros). So, I was forced to use macros, which after all is not that bad, because it also forced and garantees consistent use of keywords.

```
\def\NameKey{name}
```

So, `name` is stored once, and can be accessed by `\NameKey`, not only many times without taking string memory, but also pretty fast. This is only true in CONTEXT deepest inners, because at the user level, one does not type in macroded keys, but verbose ones: `name=hans`.

```
1: \def\Name{Hans}
    1    2   3 4 5 6 7 8

2: \def\SurName{Hagen}
    1    2    3 4 5 6 7 8 9

3: \def\FullName{Hans Hagen}
    1    2    3 4 5 6 7 8 9 11 13
```

This is one way of storing names, but when one wants to save space, the next alternative is more efficient.

```
4: \def\FullName{\Name\space\SurName}
    1    2    3   4     5      6    7
```

Counting tokens is not the way to determine this, because several types of memory are involved: the hash table, the string pool with string pointers, main memory, etc. Therefore we only really save space when more than one refer-

ence is made.

Coding keywords in macros can also be faster, especially when passing large arguments, skipping branches in conditionals and while doing certain low level lookups.

Optimizing code

One can sqeeze quite some speed from clever coding macros and using memory as efficient as possible, but when one hits the frontiers of coding itself, other methods are needed.

I started experimenting with optimization when rewriting part of the system modules. I found out that in one occasion grouping speeds up, while in another rather similar situation doing the housekeeping myself was to be prefered. Potential slow–downers are: passing long arguments, all kind of tests, especially string comparisons, list processing, and rather massive catcode changes.

Original TEX is frozen. Named E-TEX, its successor offers some basic programming features that are meant to speed up TEX as well as provide more control to macro programmers. Being rather curious, I decided to change some low level code and look to what extend E-TEX would speed up a large package like CONTEXT. (Notice that speed was not the primary target of the E-TEX project.)

Although currently most of the critical parts of CONTEXT are rather well optimized —I'm still documenting, optimizing and sometimes recoding the source— I consider the measurements to be be rather representative. It is, by the way, always an interesting dilemma: do I code for speed or for readability. It's one of the reasons why font handling routines often look rather obscure: a pretty complex font mechanism demands dirty coding to get acceptable speed.

When testing for speed, it can be tempting to put some critical code in a loop, and execute this code for instance 50.000 times. Such experiments demonstrate that individual pieces of code can be rewritten in new E-TEX primitives to run much faster. Bringing down runtime from 10 seconds to 5 seconds is fine, but in practice those fragments are not executed that many times, so the gain in normal production runs is minimal. When the protection mechanisms that CONTEXT uses for savely testing keywords are rewritten in a for E-TEX more natural way, there is even a speed penalty!

I therefore tend to conclude that it does not make much sense to recode a large macro package in E-TEX (version 2) for the sake of speed alone. This is mainly due to the fact that the critical components of CONTEXT are already coded rather efficient. Later on I will show that in one area, E-TEX beats TEX pretty well. Keep in mind that we are discussing speed and space. Much of E-TEX's new functionality goes beyond that and concerns better typography. This gives us another reason to use E-TEX.

An interesting observation is that using the new primitives `\protected` and `\ifcsname` saves about 500 hash entries in the current version of CONTEXT. However, making the specific pieces of macro code usefull for normal TEX and E-TEX, costs about 500 entries in normal TEX. Alas, that's the price original TEX users have to pay for progress.

Being parameter driven, CONTEXT does a lot of string and list processing and unfortunately TEX lacks low level support for this. When I discussed this with Taco Hoekwater, we came to the conclusion that some more straightforward support for string and list handling could speed up CONTEXT considerably, and Taco decided to extend TEX the program. That way we can present the E-TEX team with well defined and tested functionality for future versions.

When testing some first versions of Taco's binaries, I wondered if it would make sense to optimize TEX macro code in another way. To understand what I mean, I refer to a few pages back, where I introduced those hash entries and pointers. In writing macros, I try to be as clear as possible, so instead of

```
\csnamehello\endcsname
    1   23456      7
```

I code

```
\getvalue{hello}
    1    2345678
```

and not

```
\doifelse\Alpha{Beta}
    1        2   345678
```

but, at the cost of 16 bytes overhead and a two more 'lookups':

```
\doifelse{\Alpha}{Beta}
    1     2    3   456789
```

Did you notice the difference in the number of tokens used? Using a syntax highlight editor,

```
\dimen0=0pt
   1   23456
```

just looks better and more readable than

```
\dimena\z@
   1    2
```

Currently, and this is of course due to the fact that we are dealing with macros, we are also more talking of translating than of compiling. Some first experiments with an optimizer written in PERL were promising, but fearing unwanted side effects I decided to let this rest for a while.

The optimizer

And then Han The Thanh asked me to test his first version of PDF-E-TEX. This merge of two rather important developments in the TEX world: E-TEX and PDFTEX, drove me into making CONTEXT more permanently E-TEX aware as it was already pretty PDFTEX aware. While testing, I also picked up the optimization thread.

Using the timing build into TEXEXEC I found out that on a document of average complexity, the interactive MAPS bibliography, with 700 simple pages and 50 pages of cross linked indexes and lists, rewriting some core macros to use E-TEX functionality saves about 5% run time and the optimizer gives us an additional 5%: not impressive, but useful when one considers that I quite often run jobs that take an hour or more. And, apart from more efficient coding, I expect to gain another 10% in due time.

Before I will mention some characteristics of the CONTEXT optimizer, I need to give E-TEX some more credit. When I first timed the test run, I found out that E-TEX took 65% of the time the normal run needed. This proved to be due to the fact that outside E-TEX one has to fake multiple marks, and this fake can slow down a run with many color changes on a page considerably (due to lots of list manipulations). When using normal TEX, CONTEXT spends half of the run time on the 50 pages mentioned before. Because marks are used for keeping track of color, and because these pages have colored hyperlinks in multiple columns, a speed penalty is paid. Unless one heavily uses color in rather complicated documents, one will probably never notice.

When color does not cross pages, which did not really happen in this document, CONTEXT can be told to switch to local color mode. In local mode, E-TEX's gain in speed is reduced to the mentioned 5%.

Some details

Back to the optimization. Apart from a few critical files, 100 modules that are part of the distribution are optimized in four converging passes (doing it in one pass is much slower). In the process, over 7.000 lines out of 80.000 lines of macro code are optimized, in many occasions, more optimizations per line. All changes are logged and some statistics are kept. Dubious and potential dangerous situations are skipped. Of course, E-TEX optimizations are optional. Don't confuse this optimization with rewriting core macros.

Some lines are skipped. Think of macro headings and calls that use delimiters like `\box`. Optimizing macro headings is 'not done' anyway, but the next substitutions are quite legal:

- removing redundant equal signs
- using `\empty` instead of `{}`
- using `\z@` instead of 0pt
- changing 2, 4, 6 and 8 into constants in box primitives
- changing 2, 4, 6 and 8 registers into predefined ones
- using macro constants where possible
- substituting TEX keywords by macros
- changing `\getvalue` cum suis into `\csname`
- optimizing all kind of `\doif...` macros
- applying E-TEX's `\ifcsname` and `\ifdefined`
- removing redundant `{}` in arguments
- removing redundant TEX keywords

As said, sometimes substituting can be dangerous. Changing for instance

```
\expandafter\ifx\csname...\endcsname\relax
```

into E-TEX's sequence:

```
\unless\ifcsname...\endcsname
```

can lead to unwanted side effects. The pure TEX alternative creates a hash entry, that defaults to `\relax`, which is why we test for `\relax`. We just consider `\relax` to represent undefined. The second one does not create an entry. Consider for instance that at a certain moment, for instance in the process of font switching,

```
\csname...\endcsname
```

is expanded. This leads to ... being set to `\relax`. Now, when in pure TEX, we test for existance, as expected we get reported back that the control sequence does not exist. In E-TEX however, `\ifcsname` is unrelated to `\relax`, and therefore, E-TEX reports that the control sequence does exist. Even if you don't completely understand what I'm talking about, you can imagine that macros that until now work perfectly ok, fail under E-TEX.

The size of the format file (the memory dump) when optimized is currently about 75 KB less than the non–optimized file (about 2.8 MB) and some quick tests show that another similar saving is possible. It's not that much a problem to save far more bytes, but sometimes speed goes over space:

1: `\setbox0=\hbox{something}`
   1 2 3 4 5 6 7 8 9 11 13 15

2: `\setbox0\hbox{something}`
   1 2 3 4 5 6 7 8 9 11 13

3: `\setbox\boxa\hbox{something}`
   1 2 3 4 5 6 7 8 9 11 13

4: `\setboxa\hbox{something}`
   1 2 3 4 5 6 7 8 9 11 13

5: `\sethboxboxa{something}`
   1 2 3 4 5 6 7 8 9 11

The third alternative, with \boxa being \chardef'd to zero is the fastest. The last two alternatives save memory (8 bytes per macro name). It does not take much fantasy to see that the third alternative involves the two tokens \setbox and \boxa, while the fourth one takes only one (\setboxa). When resolved, this indirect reference itself takes two, therefore totalling up to three.

```
\def\setboxa{\setbox\boxa}
```

Some gain in speed in E-TEX is due to optimizing existing TEX code. I did not compare the results to original TEX, but especially saving and restoring and in some situations more clever \aftergroup handling has proved to be a welcome extension.

Conclusion

What can we conclude. First that 15 years of TEX has proven that using memory dumps makes sense. Next that for gaining some speed, developments like E-TEX make sense too. For me however the most interesting conclusion is that some sort of preprocessing makes much sense too. Because this optimization is in many respects dependant of the way the macro package is written, it is not possible to bring this into E-TEX the program. Anyhow, given this 10% speed gain, makes me very optimistic about Taco's estimations on string and list processing. It can also be a step towards a higher level of typographic programming in TEX, where more readable definitions are compiled into their lower level TEX equivalents.

Literature

- The E-TEX manual, version 2, February 1998, Peter Breitenlohner.
- E-TEX, a 100% compatible successor to TEX, Philip Taylor, EuroTEX 1995 proceedings.
- Examples: especially the CONTEXT modules syst-gen, supp-mrk, font-ini.
- Unpublished e-mails, Taco Hoekwater.

formatting

Tabulating in ConTEXt
text flow tables

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

abstract
This article describes the ConTEXt tabulate environment, which can be used to typeset tables that are part of the text flow. This mechanism differs from the TABLE based table mechanism, but recognizes the same preamble commands. It offers automatic width calculations when typesetting (multiple) paragraphs in tables and splits the tables over pages.

keywords
tables, alignment, ConTEXt

In a text, tabulated information can be included either in the text flow, fixed at the place it should appear, or it can be included at a preferred place, but given some freedom to float when there is no room. The tabulate commands discussed here take care of the fixed alternative, while the table commands are primary meant for floats. However, one is free to use whatever suits best, because none of them are limited to the cases mentioned.

While the table commands are in fact a layer around the TABLE package, the tabulate commands are written from scratch. Both have a definition preamble, and to keep things simple, the tabulate preamble keys are similar to those used in tables. We use `\NC` to as column separators, and `\NR` to go to the next row.

```
\starttabulate[|l|c|r|]
\NC this and that \NC left and right \NC here and there \NC \NR
\NC such and so   \NC up and down    \NC on and on      \NC \NR
\stoptabulate
```

this and that	left and right	here and there
such and so	up and down	on and on

The three keys mean:

`l` left aligned
`c` centered
`r` right aligned

There are also some spacing commands. These apply to both line and paragraph columns.

`i`*n* set space left
`j`*n* set space right
`k`*n* set space around

In these three keys *n* multiplies the spacing unit as set up with `\setuptabulate` (default: .5em).

```
\starttabulate[|l|k2c|r|]
\NC this and that \NC left and right \NC here and there \NC \NR
```

```
\NC such and so   \NC up and down    \NC on and on      \NC \NR
\stoptabulate
```

this and that left and right here and there
such and so up and down on and on

It is possible to specify the width of a column:

```
\starttabulate[|lw(4cm)|w(4cm)l|r|]
\NC this and that \NC left and right \NC here and there \NC \NR
\NC such and so   \NC up and down    \NC on and on      \NC \NR
\stoptabulate
```

this and that left and right here and there
such and so up and down on and on

The main reason for writing these tabulate commands, was that we wanted to typeset tables with paragraphs that would span the full available width. The related preamble keys are:

`w(`*d*`)` fixed width one liner
`p(`*d*`)` fixed width paragraph
`p` maximum width paragraph

In the next example the first column has an unknown width, the next one contains a left aligned paragraph and has a width of 4cm. The third column has 2cm width one–liner, while last paragraph column occupies the rest of the available width.

```
\starttabulate[|l|p(4cm)l|w(2cm)|p|]
...
\stoptabulate
```

A four column table with paragraph entries can be specified by saying:

```
\starttabulate[|p|p|p|p|]
...
\stoptabulate
```

Instead of retyping font switches, we can specify a font for a column. In the next table, the preamble specifies `[|lT|p|]`.

`B` **boldface**
`I` *italic*
`R` roman
`S` slanted
`T` teletype

For math there is:

`m` in line math mode
`M` display math mode

Using the `f` key, it is possible to define arbitrary font switches, for instance `f\bs`. There are some more hooks:

`f\command` font specification

`b{..}`	put before entry
`a{..}`	put after entry
`h\command`	do with entry (hook)

The next example shows how to apply a hook (`h`) and also puts something around en entry.

```
\starttabulate[|w(2cm)h\inframed|b{(}a{)}|p|]
\HC {Ugly} \NC indeed      \NC he said.     \NC \NR
\HC {Nice} \NC but useless \NC I would say. \NC \NR
\stoptabulate
```

Here `\inframed` keeps the frame within the normal line height and depth (this is a special case of `\framed`). Watch careful: hooked entries are marked with `\HC` and don't forget the braces, or whatever the hook–command expects! This example turns up as:

Ugly	(indeed)	he said.
Nice	(but useless)	I would say.

One can use the hook for specific formatting purposes, like in:

item	**quantity**
figures	
tables	
formulas	

There are three specially typeset cells. Watch the braces!

```
\def\SomeBar#1{\blackrule[width=#1em]}
\starttabulate[|l|lh\SomeBar|]
\HL
\NC \bf item \NC \bf quantity \NC \NR
\HL
\NC figures  \HC  {5}         \NC \NR
\NC tables   \HC  {8}         \NC \NR
\NC formulas \HC {12}         \NC \NR
\HL
\stoptabulate
```

Until now, we used `\NC` to go to the next column. For special purposes one can use `\EQ` to force a column separator.

```
\starttabulate
\NC equal \EQ one can change the separator by changing the \type {EQ}
              variable with the tabulate setup command \NC \NR
\NC colon \EQ by default, a colon is used, but an equal sign suits
              well too \NC \NR
\stoptabulate
```

Typeset this turns up as:

equal : one can change the separator by changing the `EQ` variable with the tabulate setup command
colon : by default, a colon is used, but an equal sign suits well too

We've seen \NC for normal entries, \EQ for entries seperated by an equal sign, and \HC for hooked ones. There is also \HQ for a hooked entry with separator. When one does not want any formatting at all, \RC and \RQ can be used.

	normal	**raw**	**hook**
equal	\EQ	\RQ	\HQ
none	\NC	\RC	\HC

This small table shows all three categories. We've got 4 centered columns, either **bold** or verbatim and two cells are aligned different. This table is coded as:

```
\starttabulate[|*{4}{cBh\type|}]
\NC           \NC normal \NC raw   \NC hook  \NC \NR
\RC \bf equal \HC {\EQ}  \HC {\RQ} \HC {\HQ} \NC \NR
\RC \bf none  \HC {\NC}  \HC {\RC} \HC {\HC} \NC \NR
\stoptabulate
```

The equal sign, or whatever else symbol is set up, can also be forced by the e key in the preamble.

e insert an equal symbol in the next column

When we have multiple columns with similar templates, we can save some typing by repeating them. We have of course to make sure that the number of |'s is ok.

```
\starttabulate[|*{6}{k1pc|}]
\NC this and that \NC left and right \NC here and there \NC
    such and so   \NC up and down    \NC on and on      \NC \NR
\stoptabulate
```

Counting the |'s, we have $1 + 6 \times 1 = 7$ of them.

this and that	left and right	here and there	such and so	up and down	on and on

A better example of automatic width calculation is given below:

tables We use the \starttable command when we are typesetting tables that are separate entities, that may float and are sort of independent of the normal text flow.

tabulate The \starttabulate command is meant for typesetting tabular text in the normal text flow. Therefore automatic width calculation, as demonstrated here, comes in handy.

This was entered as:

```
\starttabulate[|l|p|]
\NC tables   \NC We use the \type {\starttable} command when we are
                typesetting tables that are separate entities, that
                may float and are sort of independent of the normal
                text flow. \NC \NR
\NC tabulate \NC The \type {\starttabulate} command is meant for
                typesetting tabular text in the normal text flow.
                Therefore automatic width calculation, as demonstrated
```

```
                here, comes in handy. \NC \NR
\stoptabulate
```

When no template is given, `[|l|p|]` is assumed, which sometimes saves some typing. There is however a better way to save time, because one can define specific tabulate environments, like:

```
\definetabulate[Three][|lB|lS|p|]
\startThree
\NC one \NC two \NC three four five six seven eight nine ten
                    eleven twelve and so on and on and on \NC \NR
\stopThree
```

one two three four five six seven eight nine ten eleven twelve and so on and on and on

The three tabulate commands can be summarized with:

```
\definetabulate[.1.][.2.][.3.]

.1.      name
.2.      name
.3.      text
```

The first argument identifies the tabulation. The second argument is optional and identifies a related tabulation. More on that later. The last argument always defines the preamble.

```
\starttabulate[...][..,..=..,..] ... \stoptabulate

...      text
..=..    see \useexternalfigure
```

The (optional) first argument holds the preamble, and the optional second one can be used to change settings.

```
\setuptabulate[...][..,..=..,..]

...          name
unit         dimension
indenting    yes no
before       command
after        command
inner        command
EQ           text
```

The optional argument specifies a related tabulation. By setting `indenting` to `yes`, the table is indented according to the current indentation scheme. Left and right skips are always taken into account! The unit concerns the spacing as set by the spacing keys `i`, `j` and `k`. Commands assigned to `inner` are executed before the first columns is typeset.

One can add horizontal lines to a table by `\HL`. This command automatically takes care of spacing:

```
\starttabulate[|l|p|]
\HL
\NC small  \NC They say, small is beautiful.            \NC \NR
\HL
\NC medium \NC It seems that the medium is the message. \NC \NR
\HL
\NC large  \NC Large T||shirts are always sold out.     \NC \NR
\HL
\stoptabulate
```

When a page break occurs at such a horizontal rule, the rule is automatically duplicated. One can force top, mid or bottom rules with \FL, \ML and \LL.

small	They say, small is beautiful.
medium	It seems that the medium is the message.
large	Large T–shirts are always sold out.

Although the tabulate environment is primary meant for use in the text flow, it is quite legal to use it in floating tables. When used as float, the spacing around a tabulation is automatically surpressed. In the text flow however, the tabulate commands adjust themselves to the current text width and indentation.

- This means that a table created with this environment can be used within for instance an itemize.

  see this As to be expected, paragraph entries are automatically adjusted to the smaller text width.

- This small table was entered as:

```
\starttabulate
\NC see this \NC As to be expected, paragraph entries are
             automatically adjusted to the smaller text
             width. \NC \NR
\stoptabulate
```

Although tables in themselves can be used to format text in columns, occasionally using for instance an itemize in a table makes sense. The next, a bit weird, example shows this.

1. first	□□□□□ this or that	α. alpha
2. second	□□□□□ such or so	β. beta
3. third	□□□□□ here or there	γ. gamma

In such situations, packing items just looks better.

```
\starttabulate[|p(2cm)|p(5cm)|p|]
\NC \startitemize[n,packed]
    \item first \item second \item third
    \stopitemize
\NC \startitemize[packed][items=5,width=2.5em,distance=.5em]
    \its this or that \its such or so \its here or there
    \stopitemize
\NC \startitemize[g,packed,broad]
```

```
    \item alpha \item beta \item gamma
    \stopitemize
\NC\NR
\stoptabulate
```

Because the table content is read in before it is processed, there are some limitations when on–the–fly `\catcode` changes are involved. In day–to–day use however, one will not run into trouble that fast. The tabulate environment is mildly E-TEX aware, so one can expect less `\catcode` related problems when using this TEX alternative.

Tabulations can for instance be used to compose tables with numbered material, like the one below:

not to much	1.220
pretty much	5.186
totalling up to	6.406

This table is entered as:

```
\starttabulate[|l|r|]
\NC not to much     \NC          1.220  \NC \NR
\NC pretty much     \NC          5.186  \NC \NR
\NC totalling up to \NC \overbar{6.406} \NC \NR
\stoptabulate
```

Sometimes one wants to align entries in such tables by hand, and like in the TABLE based table environment, we can use ~ for this purpose. This character inserts an one digit wide space. Normally such a tie produces an unbreakable space, so changing its meaning is to be specified in the preamble.

```
\starttabulate[|l|~c|]
\NC this much \NC ~12 \NC \NR
\NC that far  \NC 185 \NC \NR
\stoptabulate
```

The next few examples show how we can define related tabulations. Actually, the main reason for programming this mechanism, originates in the wish to re–implement the legend macros.

```
\definetabulate [legend]       [|emj1|i1|mR|]
\definetabulate [legend] [two] [|emj1|emk1|i1|mR|]
\setuptabulate  [legend]       [unit=.75em,EQ={=}]
```

Now we can say things like:

```
\startlegend
\NC w \NC the width of the box  \NC pt \NR
\NC h \NC the height of the box \NC pt \NR
\NC d \NC the depth of the box  \NC pt \NR
\stoplegend
```

This simple legend turns up as:

w	=	the width of the box	pt
h	=	the height of the box	pt
d	=	the depth of the box	pt

An additional entry is possible with the `two` alternative.

```
\startlegend[two]
\NC w \NC width  \NC the width of the box  \NC pt \NR
\NC h \NC height \NC the height of the box \NC pt \NR
\NC d \NC depth  \NC the depth of the box  \NC pt \NR
\stoplegend
```

This related tabulation inherits the settings from the parent tabulation. Of course we could have defined `\startlegendtwo`, but we wanted downward compatibility with existing macros.

w = $width$ = the width of the box pt
h = $height$ = the height of the box pt
d = $depth$ = the depth of the box pt

The related fact macros are defined as:

```
\definetabulate [fact] [|R|ecmj1|i1mR|]
\setuptabulate  [fact] [unit=.75em,EQ={=}]
```

The first column is typeset in roman, the next one is separated from the first one by an equal sign, is centered, is typeset in math mode, and gets a bit more space afterwards. The last column is typeset in math mode, but inside there we switch to roman; some extra space is added in front. So:

```
\startfact
\NC width  \NC w \NC 48pt \NR
\NC height \NC h \NC  9pt \NR
\NC depth  \NC d \NC  3pt \NR
\stopfact
```

Indeed gives:

width w = 48pt
height h = 9pt
depth d = 3pt

In real life the definitions shown here also have something assigned to `inner`, which enables more compact specifications:

```
\startfact
\\ width  \\ w \\ 48pt \\
\\ height \\ h \\  9pt \\
\\ depth  \\ d \\  3pt \\
\stopfact
```

We show one last example, demonstrating the automatic paragraph width calculation. This example also shows that the last `\NC` is redundant.

```
\starttabulate[|Bl|p|Bl|]
\NC Example \NC \input tufte \NC Edward Tufte \NR
\stoptabulate
```

Example We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look in- **Edward Tufte**

to, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

As can be expected, these commands have their dutch and german counterparts:

english	dutch	german
\starttabulate	\starttabulatie	\starttabulator
\definetabulate	\definieertabulatie	\definieretabulator
\setuptabulate	\steltabulatiein	\stelletabulatorein

The preamble commands and column and row separators are independant of the interface.

Bug or Feature?
misleading font messages

Hans Hagen
Taco Hoekwater

abstract

TEX error messages and warnings are not always that clear. Sometimes confusion is due to optimizations in TEX the program. We will discuss the not always honest \the\font.

keywords
fonts, error messages

TEX is written in a time when computer resources were not as generous as in our days. This means that in TEX the program there are optimizations wherever possible. Sometimes these optimizations have unexpected side effects.

Make yourself a file saying:

```
\font\one  =cmr10
\font\two  =cmr10
\font\three=cmr10

\one   One   \message{\the\font}
\two   Two   \message{\the\font}
\three Three \message{\the\font}
```

When this file is run through TEX, you will see that in all occasions, TEX reports that font \three is used. This is due to the fact that TEX only loads a font once, which in itself is one of the reasons why preloading fonts in plain TEX can speed up font definitions in production runs (the TEXbook explains this).

The undesirable side effect is that, although the font is still accessible by the old name(s), the last name used in defining one is reported back in messages concerning for instance overfull boxes. The degree of confusion arising from this situation depends on the way the macro package names fonts. Especially when a fall back mechanism is implemented, users can start searching for the wrong causes of the problem. Imagine a message mentioning problems with typesetting text (an overfull box for example) where a sans serif font is referred to, while in fact a serif is used as fall back.

This side effect can revelate itself rather disguised in high level font mechanisms. It took both authors quite some time nailing down the origin of these confusing messages, especially because one would expect the opposite: report the old name. Finding the source of the problem was complicated by the fact that (depending on the application) it occurs independant of grouping, and therefore is pretty hard to trace. We hope that in a next release of E-TEX there will be an appropriate warning issued in the log file when this low level optimization takes place.

to, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

As can be expected, these commands have their dutch and german counterparts:

english	dutch	german
\starttabulate	\starttabulatie	\starttabulator
\definetabulate	\definieertabulatie	\definieretabulator
\setuptabulate	\steltabulatiein	\stelletabulatorein

The preamble commands and column and row separators are independant of the interface.

Bug or Feature?
misleading font messages

Hans Hagen
Taco Hoekwater

abstract

TEX error messages and warnings are not always that clear. Sometimes confusion is due to optimizations in TEX the program. We will discuss the not always honest \the\font.

keywords
fonts, error messages

TEX is written in a time when computer resources were not as generous as in our days. This means that in TEX the program there are optimizations wherever possible. Sometimes these optimizations have unexpected side effects.

Make yourself a file saying:

```
\font\one  =cmr10
\font\two  =cmr10
\font\three=cmr10

\one   One   \message{\the\font}
\two   Two   \message{\the\font}
\three Three \message{\the\font}
```

When this file is run through TEX, you will see that in all occasions, TEX reports that font \three is used. This is due to the fact that TEX only loads a font once, which in itself is one of the reasons why preloading fonts in plain TEX can speed up font definitions in production runs (the TEXbook explains this).

The undesirable side effect is that, although the font is still accessible by the old name(s), the last name used in defining one is reported back in messages concerning for instance overfull boxes. The degree of confusion arising from this situation depends on the way the macro package names fonts. Especially when a fall back mechanism is implemented, users can start searching for the wrong causes of the problem. Imagine a message mentioning problems with typesetting text (an overfull box for example) where a sans serif font is referred to, while in fact a serif is used as fall back.

This side effect can revelate itself rather disguised in high level font mechanisms. It took both authors quite some time nailing down the origin of these confusing messages, especially because one would expect the opposite: report the old name. Finding the source of the problem was complicated by the fact that (depending on the application) it occurs independant of grouping, and therefore is pretty hard to trace. We hope that in a next release of E-TEX there will be an appropriate warning issued in the log file when this low level optimization takes place.

column

Don’t give authors the class files!

Kaveh Bazargan
Focal Image Ltd
kaveh@focal.demon.co.uk

Publishers are now used to taking TEX or LATEX files from authors and using them in publishing books and journals. It has become a standard procedure now for LATEX class files to be distributed to interested authors. Publishers will subcontract TEXies to produce these to their particular styles. It is, in some cases at least, a bit like web sites. They want them because everyone else has them – as much PR as anything else.

Based on being involved in the typesetting of tens of thousands of LATEX pages, I want to argue against giving authors these class files. Here’s why:

- Not all class files can be used by the average author. This is not necessarily because they are bad class files, but they need careful use. They need some manual tweaking in the final stages, and this is not always trivial. The trick is to do this with minimal effect on the structure of the source code, so the code can be reused. The author may attempt to do this, but will invariably come up with an inelegant solution. He may even edit the class file in an effort to fix things.
- Authors will get the impression that they are producing camera–ready copy. In a minority of cases they will be able to produce CRC successfully, but again, in an effort to get the pages looking right they will produce ugly code to to generate reasonable output.
- Most class files are written by LATEX specialists for their own use in–house, for example for typesetting material sent in by publishers. (I wish we could banish that term – typesetting!) They sometimes have dirty fixes which they have never had time to clean up, but which nevertheless produce the correct results.
- Commercial fonts cannot legally be copied and sent to authors. So the publisher would have to have multiple copies of the fonts, for multiple platforms, in stock. Alternatively the style would have to be limited to public domain fonts.
- Presently, most publishers do not request an archive of the LATEX source code. They are mostly interested in POSTSCRIPT, PDF, and sometimes SGML headers. They are not worried about the structure of the source code. But as output becomes increasingly electronic, the structure of the code will be important, so that SGML or XML files can be produced with minimal work.

So what is the solution? Well, I think publishers should be more ‘pro–active’ in dealing with authors. In other words, they should change tactics and actively try to come up with a system that works. Authors will respond to well set out instructions, especially if they think it will improve the efficiency of processing their files.

I think that publishers should send out an ‘author kit’ to prospective authors, and active encourage the use of LATEX, rather than just say they will accept the files. Here is what I think should be in the author kit:

- **A ‘generic’ class file.**
  This class file would probably be based on the standard LATEX Article or Book class files, but with modifications. It would use standard Computer Modern fonts, and would be designed with generous line and page breaking glue, so as to avoid overful boxes wherever possible. In short the class file would be designed to make it easier for the author to enter his text. The generic class file would contain all the environments necessary, as envisaged by the publisher, but the visual output would look nothing like the final typeset pages.
- **A user–friendly instruction manual.**
  Written in the spirit of Lamport’s standard manual, a small user guide can gently encourage the author to produce clean, structured text. Any special instructions should be written into a `readme` file. I think that a well written set of instructions will raise the profile of the publishers considerably.
- **A fully working sample file.**
  This is probably the easiest way to commnunicate the workings of the class file. A simple document with examples of each environment available will help further to guide authors in the right direction.
- **Lamport’s manual.**
  Why not? I would say that for a prospective book author, this is not a major expenditure, and will pay off in goodwill.

Here are the advantages of taking this generic file approach:

- Publishers are free to implement more ambitious styles, getting away from the usual 'TEX' look. For example the designers can have a freer hand in choosing fonts, using shading, colour, PostScript tricks, MetaPost etc. All this without having to update the author class files at all.
- Authors can concentrate on what they should, namely the content of their paper – in the original spirit of TEX.
- Support for authors becomes a simple matter, being confined to one file.

I think the procedure outlined means less work for the author and the publisher, and more accurate typesetting. Typesetters would rather work with clean, structured LATEX files than files that look almost like the final product, but which have bad code. Here's another advantage: if the procedures work well, then the typesetter can return to the author the LATEX files used in the final typesetting of the book, by removing any visual codes used in the final stages of pagination. For book authors, the files can be used in preparing the next edition. This very rarely occurs at present. When conventional typesetting systems are used, the native file can only be manipulated by the same (usually expensive) typesetting program.

boekbespreking

Robin Williams over typografie

Een dame geeft raad

Frans Goddijn
fg@fgbbs.iaf.nl

abstract

Tijdens de heuglijke NTG-dag in het vorstelijke Leuven vond ik op de boekentafel van Addison-Wesley twee leuk uitziende boeken over typografie en ontwerp. Mooie, knalgele boekjes die even onderhoudend als leerzaam bleken, geschreven door Robin Williams: *The Non-designers's Type Book* en *The Non-designers's Design Book*. Ik sla er één voor u open.

keywords

typografie, letters, ontwerp, expert

Geel?

Is geel een signaalkleur waarmee hedendaagse typografen aan elkaar laten zien dat ze 'hip' zijn? Het lijkt er wel op. Ik heb niet erg veel boeken over typografie en ontwerp, maar bijna allemaal vlaggen ze felgeel. De 'Tekstwijzer' van K.F. Treebus (SDU uitgeverij) is nog degelijk grijs, evenals het klassieke 'Boek over het maken van boeken' door de *eminence grise* Huib van Krimpen (Gaade Uitgevers) maar het loodzware kleurige kijkboek 'TYPO' over de beste typografen van de wereld (uitg. Könemann) heeft al een paar spetterende vlakken geel op de cover, de langwerpige, vuistdikke encyclopedie 'Fontbook' van FontShop is net zo kanariegeel als mijn andere FontShop uitgaven, het mooie typografieboek 'LetterFontein' (uitg. Fontana) heeft in geel het woord 'letter' op rug en front, Rookledge's 'International handbook of typedesigners' is okergeel en dezelfde uitgeverij gaf hun 'International TypeFinder' gele letters op zwarte achtergrond. Zelfs Joyce Irene Whalley's 'Writing implements & accessories (from the roman stylus to the typewriter)' is ooit cadmiumgeel geweest, voordat het verbleekte.

Geen wonder dus dat Robin Williams geel koos voor de door haar zelf geschreven en zelf vormgegeven boeken 'The non-designer's Type book' en 'The non-designers's Design book'.

No more cats

Er zijn zoveel Engelstaligen dat er meerdere dezelfde naam hebben. Zo las ik onlangs dat Phil Taylor een groot dartskampioen is. Dat zal wel een andere Phil Taylor zijn dan 'die van ons', de paardrijdende typograaf en TEX guru.

En nee, deze Robin Williams is niet *de* Hollywoodacteur Robin Williams, en ook niet de brutale Engelse zanger. Deze Robin Williams is een dame, die achterop haar boeken verschijnt met portretfoto's die rond de vorige eeuwwisseling lijken te zijn gemaakt. Mondain en frivool.

Vandaag de dag leeft een Amerikaan niet als er niet over zijn of haar privé-leven kan worden geroddeld en Williams doet haar best om de snuffelaars naar details enigszins tegemoet te komen. Haar boek over typografie heeft een opdracht aan 'Allan Haley, with graceful appreciation for inspiring, educating and befriending me'. Achterin het boek vertelt ze 'About the author' dat ze vlakbij Sante Fé, New Mexico woont, 'still a single mom of three kids, three dogs, but no more cats.'

Dat klinkt veelzeggend: 'but no more cats'. Het voorgaande boek over design was nog opgedragen aan 'Carmen Carr, my comrade in Design, my friend in Life. — with great love, R.' Toen woonde ze met '... three kids, three dogs, a cat, [...] and lots of wild rabbits.' Het katje is weg, en Robin hoeft niet zo nodig een nieuwe.

Typografie: training van het oog

Williams maakt van de argeloze tekstverwerkende computergebruiker een beginnende typograaf en allereerst wijst ze ons op een aantal typografische details waardoor je in een oogopslag het verschil ziet tussen dom tikwerk en beginnend typesetten. De TEX gebruiker herkent de eerste al snel: een passende lettergrootte, een passende regelbreedte, de correcte "aanhalingstekens" in plaats van het `"`-teken, het gebruik van de juiste hoeveelheid spatie in ... in plaats van ... en het verschil in breedte van afbreekstreepje, gedachtestreepje en het streepje tussen twee jaartallen. Door de voorbeelden in het boek ging ik nadenken over de gedachtestreep in TEX en een discussie ervan op `TEX-NL` leverde een mooie nieuwe macro op waarmee dit streepje — in *mijn* ogen — een mooier en duidelijker verschijning kreeg.

```
\def\denk{\leavevmode%
 \thinspace\penalty0
          \hbox{--\kern -.25em--}
 \thinspace\penalty0}
```

Een ander licht dat me door het voorwoord van Williams' boek al opging was het verschil tussen 'echte' small caps en de namaak small caps die door de computer kunnen worden gegenereerd. Bij nep-small caps is er steeds een verschil in donkerheid van de letters, die eerst niet zo opvalt, maar steeds meer gaat tegenstaan naarmate je je er meer van bewust wordt. Ga zelf eens na welke letter precies wordt gebruikt als je `\textsc` kiest. Het gebruik van ligaturen gaat net zo: in TEX krijg je ze meestal cadeau, en je begint ze pas bewust te zien als ze een keer zijn weggevallen. Daarna begin je je te ergeren aan teksten waarin ligaturen niet, of slecht worden gebruikt. De f en de i botsen dan op elkaar in het woord fiets of de ffi gaan ongemakkelijk in polonaise in het woord efficiënt.

Geschiedenis van typografie

De stoet letters vanaf het begin van de boekdrukkunst is al op menig NTG-dag tijdens een lezing via sheets langs de plaat van de overheadprojector gegleden. In Leuven nog, waar Thierry Boucher ons weidse letterlandschappen toonde van allerlei aard, ons de keuze latend van 'you like *it*... you no like *it*... pfft!'

Ook Williams behandelt deze onmisbare stof en ze doet dat kort en bondig, voor de lezer die alles wil weten maar wel snel graag en niet te moelijk. In het boek worden meer dan 115 fonts gebruikt als voorbeelden van lettersoorten (oldstyle, modern, slab serif, sans serif, fringe, scripts en decorative) die voor verschillende doeleinden kunnen worden ingezet.

Readable & legible

Ik weet niet of andere Engelstalige typografen het subtiele verschil tussen de woorden *readable* en *legible* op dezelfde manier hanteren, maar Williams creëert er een handvat mee om het verschil duidelijk te maken tussen koppen en broodtekst. De *letter* van de broodtekst van een pagina moet zo onopvallend mogelijk zijn om optimaal leesbaar te zijn. De lezer glijdt met de ogen over de tekst en haalt zo *informatie* binnen, een gedachtestroom van de auteur. De letter waarmee de gedachten worden overgedragen moet zelf niet opvallen. Als de lezer denkt: 'hee, wat een geinige q staat daar, en wat een aanstellerige griekse y zit er in dit fontje', dan is er iets foutgegaan in de keuze van de letter. Met *readable* bedoelt Williams een *leesbare* tekst die je gedachten influistert zonder zelf een rol te spelen. Een *zichtbare* tekst daarentegen wil opvallen. Het is een kopje, een kreet die allereerst gezien wil worden en dan pas begrepen. Deze *legible* letter mag een contrast vormen met de broodtekst, graag zelfs. Daarom: bijvoorbeeld een schreefloze letter voor titels boven tekstblokken in een schreefletter.

Met voorbeelden laat Williams zien waarom je in een broodtekst niet zonder meer de schreefletter kunt vervangen door een even grote schreefloze. Ga je over op een schreefloze letter, dan moet je tegelijk de regelbreedte verminderen, de regelafstand vergroten en een iets kleiner type kiezen. Ook toont ze hoe al te opvallende letters in een titel blikvangers worden zonder dat de betekenis van het woord nog valt te ontcijferen.

Woordspatie en letterspatie

'Onze' TEX is vrij goed in het uitvullen van regels en TEX beperkt zich daarin doorgaans tot het behoedzaam rekken van spatie tussen woorden. Misschien daarom struikelt mijn oog direct zodra er met spaties tussen letters wordt gerotzooid. Williams kent TEX niet, ze gebruikt PageMaker 6.5 en ze zorgt regelmatig voor een 'betere' uitvulling van een alinea door spatie tussen letters én woorden te variëren. Ik vind dat gruwlelijk en dat mag, want nu ons oog enigszins is geschoold, mogen we ook eigenwijzer worden: 'trust your eyes. If it looks like the words are too close together, they are. If it looks like the letters are too far apart, they are.'

Als je je afvraagt *hoe* je kunt variëren in letterafstanden, moet Willams lachen: 'Ha! read the manual.'

Wees geen watje!

Hoewel Williams de geldende regels van goede smaak stuk voor stuk behandelt, wil ze daarmee niet zeggen dat er niet tegen mag worden gezondigd. Integendeel, alleen: als je het doet, doe het dan opvallend fout, is haar devies. Een beetje naast de pot piesen is knoeien, er straal naast spuiten is kunst, zo mag ik het misschien vertalen. Robin Williams slaakt hier kreetjes van opwinding: 'It's okay to break the rules, but break them with gusto! [...] Don't be a wimp! [...] Oh, the possibilities are endless and exciting!'

Voor ons TEX-users is het leven echter niet zo makkelijk. Als je niets extra's doet, krijg je van TEX een redelijk uitgevulde tekst in een wat te grote letter met iets te krullerige cursieven. Het kost al moeite om fraaie typografie te produceren maar het vergt grote kracht en diepgaande TEX-kennis om uit de band te springen zonder dat je verdwaalt in foutmeldingen. En dan geef ik het al gauw op.

TEX is zo braaf en voorzichtig! Stel dat ik met een hoofdstuknummer heel groot in lichtgrijs zo wil schuiven dat het zichtbaar én leesbaar is, en dat ik een tussenkopje in witte letters op een zwarte achtergrond in een rechthoekje de tekst wil binnenschuiven zodat de tekst erom-

heen loopt... dat zijn dingen die met de handleiding op schoot met een pakket als PageMaker wel te doen zijn. In TEX is het óf doodgemakkelijk, als Hans Hagen het toevallig in ConTeXt heeft ingebouwd (wat meestal het geval is), óf hels moeilijk, bijvoorbeeld als het nog niet in de ConTeXt manual staat...

Kleur in zwart-wit

In een paar korte hoofdstukken laat Williams de lezer proeven van *kleur*. Tenminste, datgene wat drukkers en typografen kleur noemen. Stel dat binnen de alinea's op deze pagina verschillende lettertypes waren gebruikt, in romein, cursief en vet. Doordat bij het printen die ene vette alinea meer inkt krijgt dan de andere, oogt dat vlak van een afstand donkerder dan de overige, en de andere vlakken zien er ook anders getint uit. In een roman probeer je alle pagina's even grijs te krijgen, maar in andere documenten kun je bewust omgaan met die verschillen en daarmee een duidelijke visuele structuur aanbrengen op de pagina. Een korte samenvatting zou dan niet per se bovenaan hoeven te staan, het kan ook als een klein apart vlak binnen de rest van de tekst worden gezet, zolang de 'kleur' van dat tekstblok maar overduidelijk maakt dat het hier om een apart eilandje met informatie gaat, aantrekkelijk neergezet tussen de andere tekst-elementen. Ook hier moet ik zeggen dat ik verlekkerd naar de voorbeelden kijk, beseffend dat ik er vanuit TEX niets mee kan. Zoals onze kat die in de vensterbank likkebaardend miauwt naar het merel dat onbereikbaar dichtbij door de tuin hipt.

Winkelen

Kooplustig gemaakt door boeken over typografie heb ik weleens de neiging om een paar fonts te bestellen. Gewoon, 'doe mij maar een paar fonts!' Alleen, de onvermijdelijke vraag is dan 'en welke mogen het wezen?' — dat is dan wel irritant, want de winkelier wil me best wat verkopen, het aanbod is er, de prijzen zijn laag, maar het komt zelden tot een koop. Waarom? Ik kan geen keuze maken. Ik wil een font dat niet opvalt, en dat toch zo mooi is dat kenners zullen zeggen: 'Wat heb *jij* een mooi fontje!' Ook moet het een hele familie zijn, compleet met *echte* small caps en oldstyle cijfers in meerdere expert sets, liefst met 'swash' letters erbij voor als een letter in een kopje eens extra de blits mag maken. Maar stel dat zo'n font er is, en dat is er zeker, wat doe ik dan als ik mijn credit card nummer heb gegeven en als ik met de teruggekregen codes de bronbestanden heb kunnen krabben van de cd-roms die ik van de diverse fontbibliotheken in huis heb gehaald?

Als hulp bij het eerste probleem schreef Williams een hoofdstuk over fontkeuze. Daarbij komt ze met verrassende overwegingen.

Bijvoorbeeld de benadering van de andere kant van het productieproces: op welke resolutie zullen de fonts worden gebruikt? Als je de letter gebruikt voor faxen, zijn er andere eisen te stellen dan als je er een boek mee maakt dat uiteindelijk op zeer hoge resolutie wordt gedrukt. In het boek kun je letters met dunne lijnen wel kwijt, door een fax kunnen die subtiliteiten verloren gaan. Op wat voor papier wordt gedrukt? Als het 'gaatje' van de e klein is, en je print met inkt op grof papier, dan vlekt de boel dicht.

Wil je een studieboek, een roman of een kinderboek maken? Is het van belang dat de letter 'zuinig is met papier'? Een boek in times gezet zal beduidend minder pagina's nodig hebben dan hetzelfde boek in palatino, bijvoorbeeld.

Als je een broodletter hebt, vind je daar een passende, op de goede manier contrasterende letter bij voor de titels?

Stuk voor stuk eye-openers. Alleen, als je dan met een zak vol letters 'de winkel' uitloopt, zit je als TEX-user nog met de vraag hoe je dat hele typografische vuurwerkpakket het beste in je TEX kunt steken. Uit angst me een rund te voelen stel ik daarom de aanschaf vaak even uit, tot een nationale TEX-guru per ongeluk aanbiedt me 'wel even te helpen'... thanks Taco!

Zo heb ik ooit, voor ik Taco Hoekwater kende, de hele familie Van Dijck aangeschaft, een leuke fontfamilie met alleraardigste telgen. Alleen de vette zoon des huizes is net niet vet genoeg en de cursieve dochter is wel heel erg apart, eentje met zeer geaffecteerde maniertjes. Alles bij elkaar een leuk stel, maar niet geschikt voor bijeenkomsten waar ernstige teksten samenkomen en net te deftig voor vrolijke liedjes. Nog steeds moet ik daar een vet pleegkind bij zoeken en een wat minder wild cursiefje.

Dertien doodzonden

In het hoofdstuk 'Telltale signs of desktop publishing' behandelt Williams een aantal eigenschappen waarmee de lompe boeren van de typografie, de nouveau riche van de letterplakkers, de would-be vormgevers zich verraden. En, ik moet het hier eerlijk bekennen, ik ben geschrokken!

Zo wordt uitgelegd dat *niemand* meer Helvetica gebruikt. Helvetica is lelijk, dom, passé, een smerig en goedkoop residu uit de zestiger jaren. Nu het ook nog gratis zit bij de meeste printers wil geen enkele typograaf er nog dood mee worden gevonden.

En ik gebruik nog helvetica.

Ook vertelt Williams dat een grijze achtergrond van de tekst binnen een kader echt en helemaal 'fout' is. En als dat kadertje ook nog ronde hoeken heeft, is het helemaal tijd om een teiltje te pakken.

Oef, in mijn briefstijl heb ik de adressering in zo'n kader. Met grijze achtergrond nog wel.

Gelukkig bezondig ik me niet aan de andere elf mogelijke flaters die de kluns ontmaskeren. Bij dit hoofdstuk

voelde ik me wel betrapt, ongeveer zoals ik laatst, op een keurig feestje, tijdens mijn tweede glas wijn merkte dat ik moeite kreeg met het woord ‘republikein’. Replu, Repubi, replubli... Ik zal in het vervolg beter oppassen. Alleen al om dit hoofdstuk is het boek zijn geld ward.

‘...You can put a cat in the oven, but that don’t make it a biscuit’, is het motto voorin het boek. Kennelijk heeft Robin Williams de kat niet opgegeten.

Robin Williams
The Non-designers’s Type Book
Peachpit Press, Berkeley california
ISBN 0-201-35367-9

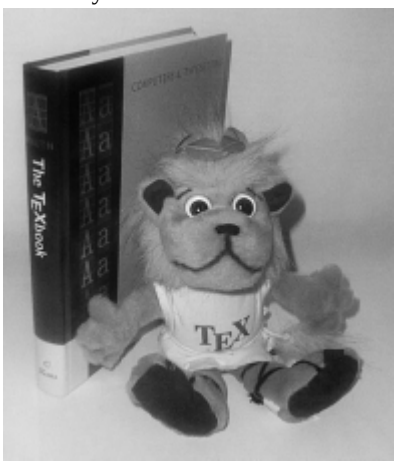
Ook verkrijgbaar:
Robin Williams
The Non-designers’s Design Book
Peachpit Press, Berkeley california
ISBN 1-56609-159-4

TEX Merchandising

Martin Schröder
Martin.Schroeder@ACM.org

The “Independent TEXnical Working Group on TEX Merchandising” (ITWGTM) proudly presents its first results: the one and only TEX lion as soft “ware” and as a pin.

The teddy:



The pin:

The teddy is available in Germany for DM 55 from
Liebscher & Partner, Am St. Niclas Schacht 13, D-09599 Freiberg/Sachsen, Tel. (+49) 03731 / 78 13 86, Fax (+49) 03731 / 78 13 77, E-Mail info@freibergnet.de, http://www.freibergnet.de
TEXies outside of Germany should contact their local TEX user groups to save handling charges by combining orders.
The pin is avaiable for DM 3 either from your local TEX user group or directly from the ITWGTM; contact me if you have any problems.
You can learn more about the ITWGTM at its homepage at http://home.pages.de/~tex-merchandising/.

graphic design
Typography to a purpose

Siep Kroonenberg
Kluwer Academic Publishers
Dordrecht
siepo@cybercomm.nl

This paper shows some real-life examples of typography, some good, some not so good. We shall have a look at what is on the page and speculate about what the publisher or designer is trying to accomplish.

Fiction

Most fiction publications consist simply of text. Good typography means typography that does not get in the way. Our two examples are a collection of short stories: 'Damon Runyon Favorites' (figure 1) and a novel 'Sybil' by Benjamin Disraeli (figure 2).

The Runyon book looks a bit sloppy because of loose word spacing, and the very narrow margins make the page look cramped. The use of page headers doesn't help, although the right page headers do contain useful information, *viz.* the titles of the stories. What the picture doesn't tell is that the printing is quite good, that the paper has a pleasant supple feel and that the cover has a nice atmospheric illustration.

The Sybil book is much more tightly designed. Margins are adequate but certainly not extravagant. There are no page headers at all, but the page looks fine without them. Page numbers are centered. From a strictly practical point of view, page numbers at the outer edge ought to be better, but I doubt whether the page would then look as good. Compared to the Runyon book, the type size is smaller, and so a more open look is combined with more text on the page – but some people are going to need their glasses...

Neither book starts a new page for a new chapter or story, and neither book uses bold type or a second typeface. For fiction books such as these, such low-key design is fine.

Reference publications

Other than in novels, you really want page headers in a dictionary (figure 3). This is one of several features to let you find information quickly. Others are outdented boldened names of entries. In more recent dictionaries you might see a contrasting typeface here.

Compare the typographic parameters with those of the two works of fiction: there, too, space was saved, but type had to remain comfortable to read even for long stretches. Here, you won't be reading more than few lines at a time and therefore smaller and narrower type is still acceptable.

A more frivolous representative from this category is 'The

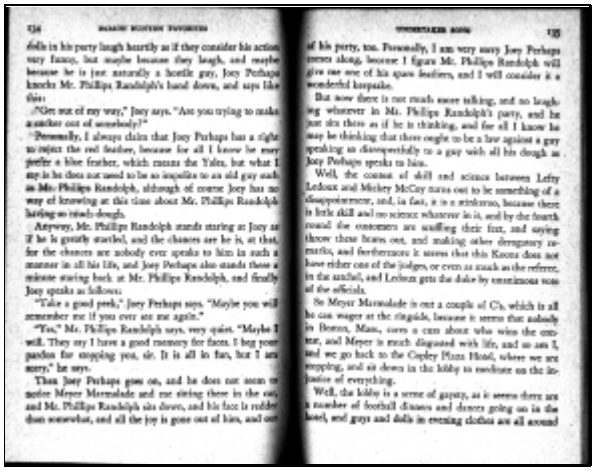
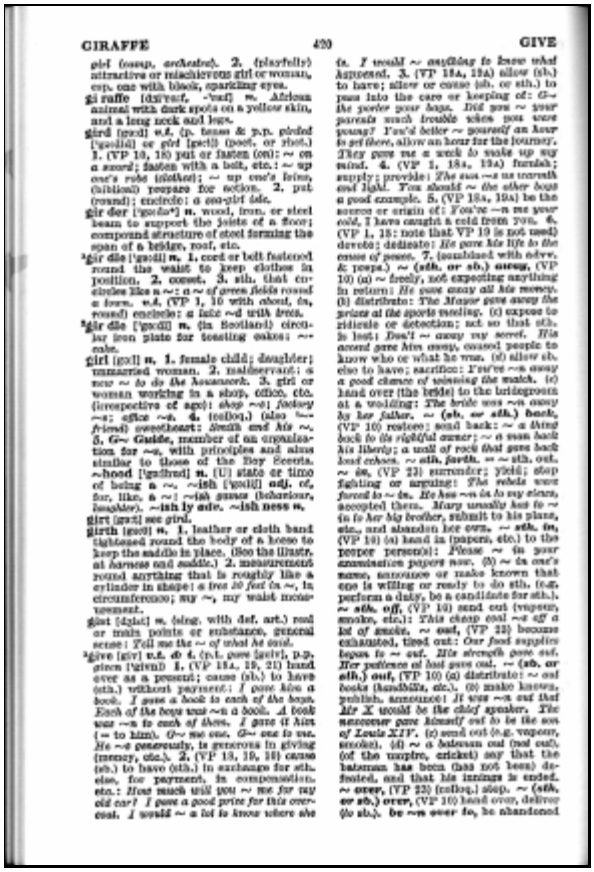


Figure 1. *Damon Runyon Favorites,* Pocket Book Editions, 1942. Short stories.

Figure 2. *Sybil,* Benjamin Disraeli, Penguin, 1954



GIRAFFE 470 GIVE

gi raffe [dʒi'rɑːf, -'ræf] *n.* African animal with dark spots on a yellow skin, and a long neck and legs.

gird [gɜːd] *v.t.* (p. tense & p.p. girded ['gɜːdid] or girt [gɜːt] (poet. or rhet.) 1. (VP 19, 18) put or fasten (on): ~ on a sword; fasten with a belt, etc.: ~ up one's robe (clothes); ~ up one's loins, (biblical) prepare for action. 2. put (round); encircle: a sea-girt isle.

gir der ['gɜːdə*] *n.* wood, iron, or steel beam to support the joists of a floor; compound structure of steel forming the span of a bridge, roof, etc.

¹gir dle ['gɜːdl] *n.* 1. cord or belt fastened round the waist to keep clothes in position. 2. corset. 3. rib. that encircles like a ~: a ~ of green fields round a town. *v.t.* (VP 1, 16 with about, round) encircle: a lake ~d with trees.

²gir dle ['gɜːdl] *n.* (in Scotland) circular iron plate for toasting cakes: ~-cake.

girl [gɜːl] *n.* 1. female child; daughter; unmarried woman. 2. maidservant: a new ~ to do the housework. 3. girl or woman working in a shop, office, etc. (irrespective of age): shop ~s; factory ~s; office ~s. 4. (colloq.) (also 'my ~) friend: sweetheart: Smith and his ~. 5. G~ Guide, member of an organization for ~s, with principles and aims similar to those of the Boy Scouts. ~hood ['gɜːlhud] *n.* (U) state or time of being a ~. ~ish ['gɜːliʃ] *adj.* of, for, like, a ~: ~ish games (behaviour, laughter). ~ish ly *adv.* ~ish ness *n.*

girt [gɜːt] see gird.

girth [gɜːθ] *n.* 1. leather or cloth band tightened round the body of a horse to keep the saddle in place. (See the illustr. at harness and saddle.) 2. measurement round anything that is roughly like a cylinder in shape: a tree 20 feet in ~, in circumference; my ~, my waist measurement.

gist [dʒist] *n.* (sing. with def. art.) real or main points or substance, general sense: Tell me the ~ of what he said.

²give [gɪv] *v.t.* & *i.* (p.t. gave [geɪv], p.p. given ['gɪvn]) 1. (VP 18a, 19, 21) hand over as a present; cause (sb.) to have (sth.) without payment: I gave him a book. I gave a book to each of the boys. Each of the boys was ~n a book. A book was ~n to each of them. I gave it him (= to him). G~ me one. G~ one to me. He was ~n generously, is generous in giving (money, etc.). 2. (VP 13, 19, 18) cause (sb.) to have (sth.) in exchange for sth. else, for payment, in compensation, etc.: How much will you ~ me for my old car? I gave a good price for this overcoat. I would ~ a lot to know where she is. I would ~ anything to know what happened. 3. (VP 18a, 19a) allow (sb.) to have; allow or cause (sb. or sth.) to pass into the care or keeping of: G~ the feeder your hops. Did you ~ your parents much trouble when you were young? You'd better ~ yourself an hour to get there, allow an hour for the journey. They gave me a week to make up my mind. 4. (VP 1, 18a, 19a) furnish; supply; provide: The sun ~s us warmth and light. You should ~ the other boys a good example. 5. (VP 18a, 19a) be the source or origin of: You've ~n me your cold, I have caught a cold from you. 6. (VP 1, 18: note that VP 19 is not used) devote; dedicate: He gave his life to the cause of peace. 7. (combined with advv. & preps.) ~ (sth. or sb.) away, (VP 10) (a) ~ freely, not expecting anything in return: He gave away all his money. (b) distribute: The Mayor gave away the prizes at the sports meeting. (c) expose to ridicule or detection; act so that sth. is lost: Don't ~ away my secret. His accent gave him away, caused people to know who or what he was. (d) allow sb. else to have; sacrifice: I've/'ve ~n away a good chance of winning the match. (e) hand over (the bride) to the bridegroom at a wedding: The bride was ~n away by her father. ~ (sb. or sth.) back, (VP 10) restore; send back: ~ a thing back to its rightful owner; ~ a man back his liberty; a wall of rock that gave back loud echoes. ~ oneself, ~ o. up, (VP 23) surrender; yield; stop fighting or arguing: The rebels were forced to ~ in. He has ~n in to my views, accepted them. Mary usually has to ~ in to her big brother, submit to his plans, etc., and abandon her own. ~ sth. in, (VP 18) (a) hand in (papers, etc.) to the proper person(s): Please ~ in your examination papers now. (b) ~ one's name, announce or make known that one is willing or ready to do sth. (e.g. perform a duty, be a candidate for sth.). ~ sth. off, (VP 10) send out (vapour, smoke, etc.): This cheap coal ~s off a lot of smoke. ~ out, (VP 23) become exhausted, tired out: Our food supplies began to ~ out. His strength gave out. Her patience at last gave out. ~ (sth. or sth.) out, (VP 10) (a) distribute: ~ out books (handbills, etc.). (b) make known, publish, announce: It was ~n out that Mr X would be the chief speaker. The newspaper gave himself out to be the son of Louis XIV. (c) send out (e.g. vapour, smoke). (d) ~ a batsman out (not out), (of the umpire, cricket) say that the batsman has been (has not been) dismissed, and that his innings is ended. ~ over, (VP 23) (colloq.) stop. ~ (sth. or sb.) over, (VP 10) hand over, deliver (to sb.). be ~n over to, be abandoned

Figure 3. *Advanced learners dictionary of current English*, Oxford University Press, 1963

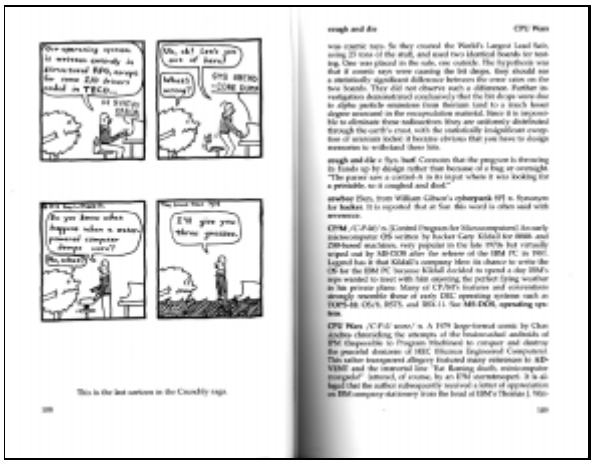


Figure 4. *The new hacker's dictionary*, Eric Raymond, MIT Press, 1991

new hacker's dictionary' (figure 4). This is much closer to conventional book typography, and of course its real purpose is entertainment, not reference. The cartoon, by Guy L. Steele, demonstrates that there is a place for amateurism.

This book is typeset with Tex[info].

Catalogues and advertising

Price lists (see figure 5) also often squeeze a lot of type in a small space. Here, too, headings should help you to get to the right spot quickly. More conspicuous though is the decoration in this example: a circuit board as general background, shaded type for headers, and shapes with graduated fills. This is certainly not done for the sake of legibility; presumably it has something to do with the image the advertiser is trying to project.

Another example, figure 6, shows more variation. This is not a magazine ad but a page from a standalone publication. Running text is set from two typeface families: one serifed and one sans-serif. The serifed text seems intended

Figure 5. Cirkit ad, from an English computer magazine, 1998

as the main story but this is not at all clear.

The page has been jazzed up in various ways: objects and price tags placed at angle; drop shadows; objects sticking out of their frame. The line drawings in the middle stand out because their style is so different from the photographic illustrations. Here, too, we find graduated backgrounds.

Is this good design? Frankly, I don't know. If I were shopping for a PCMCIA modem, I would have preferred long lists of all the PCMCIA modem cards in stock, complete with prices and specifications. If I were the kind of person who drools over hardware, I wouldn't need the specifications because I knew them by heart. If I were an impulse buyer I might fall in love with the 'bliksemse stekker' while shopping for that PCMCIA modem, so it would be a good thing (from the advertiser's point of view) that the bliksemse stekker was on the same page.

By way of contrast, here a couple of publisher's catalogues. The first one, figure 7 is from O'Reilly and contains computer books. Once more, there is real text to read: the title in bold sans and the publishing data in italic are followed by a longish descriptive text.

Figure 6. 06 catalogue, november 1998

In spite of all the text, it is quite an airy page. This is due to the large amount of white space, the alteration in the placement of the book covers, and the lack of horizontal or vertical justification. The animal covers certainly help.

Figure 7. Catalogue of O'Reilly and Associates, 1998

Figure 8. Catalogue of Ellessy, 1998

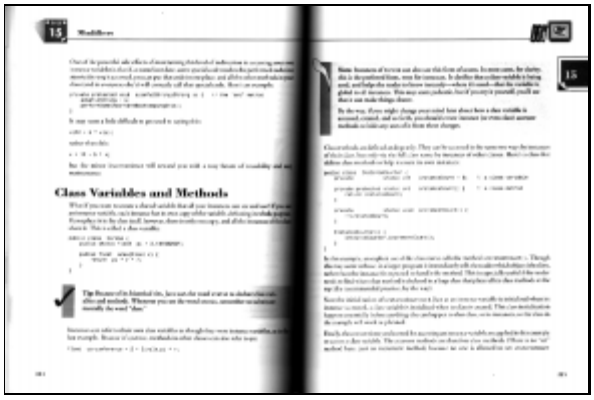


Figure 9. *Teach yourself Java in 21 days*, Laura Lemay and Charles Perkins, SAMS, 1996

The other one, figure 8, one of my own, is for a small publisher Ellessy. The author's name rather than the title is the most prominent item. The typeface is Officina Serif, which is readable enough here, but which you wouldn't want to use for a novel. Space considerations forced me to place some text alongside the cover on the righthand page. Such narrow columns of type aren't pretty and aren't easy to read.

Computer books

The shelves of bookstores are crammed with fat and ugly computer books. A quick inspection will tell you that in most cases the bloat comes from both the writing and the designing. I failed to come up with a really awful example from our own collection, since we had recently given all our unwanted books to a 'kringloop' second-hand goods store. So the worst offender I could come up with is 'Teach yourself Java in 21 days' (figure 9) The selected spread shows the abundant use of graphic symbols and ornaments and of shaded boxes. This makes it only mildly awful, it can get much worse; check for yourself at your local bookseller. The monospaced font for code is actually rather nice.

The manual of SuSE Linux (figure 10) is produced with LaTeX 2_ε, as stated on the colophon page. It also suffers from overdecoration: lots of shadowed textboxes and danger signs, boldened text and sculpted keys. This spread doesn't show it, but the makers have not been overly careful about typographic niceties such as staying within the margins or not having a headline at the bottom of the page. The English can be pretty bad, sometimes to the point of incomprehensibility. In short, the kind of production that gives TeX a bad name.

They are no longer upstarts and I think they should shed their amateurish look and hire a real designer. Nevertheless, the ugliness is rather endearing. Also, it is a very useful book for someone trying to set up a Linux system.

For a large part of its catalogue, O'Reilly sticks to its well-known formula of an animal woodcut on the cover, and the inside typeset in ITC Garamond in an unadorned style, with little variation from volume to volume; figure 11 is from 'Running Linux'. Standard for the O'Reilly books are unindented paragraphs separated by vertical whitespace, the headings in bold-italic and the indented code fragments in Courier. In this case there is an enlarged left margin for symbols such as bombs (danger) or book symbols to mark crossreferences.

O'Reilly's Unix books are almost invariably typeset from SGML via groff, a batch tool. The colophon doesn't tell how much manual intervention was required. Their

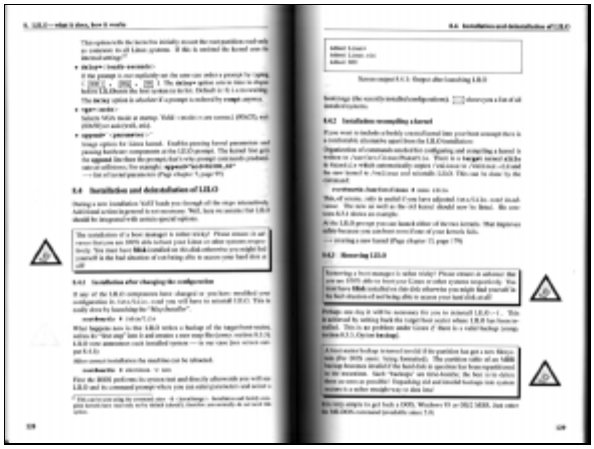
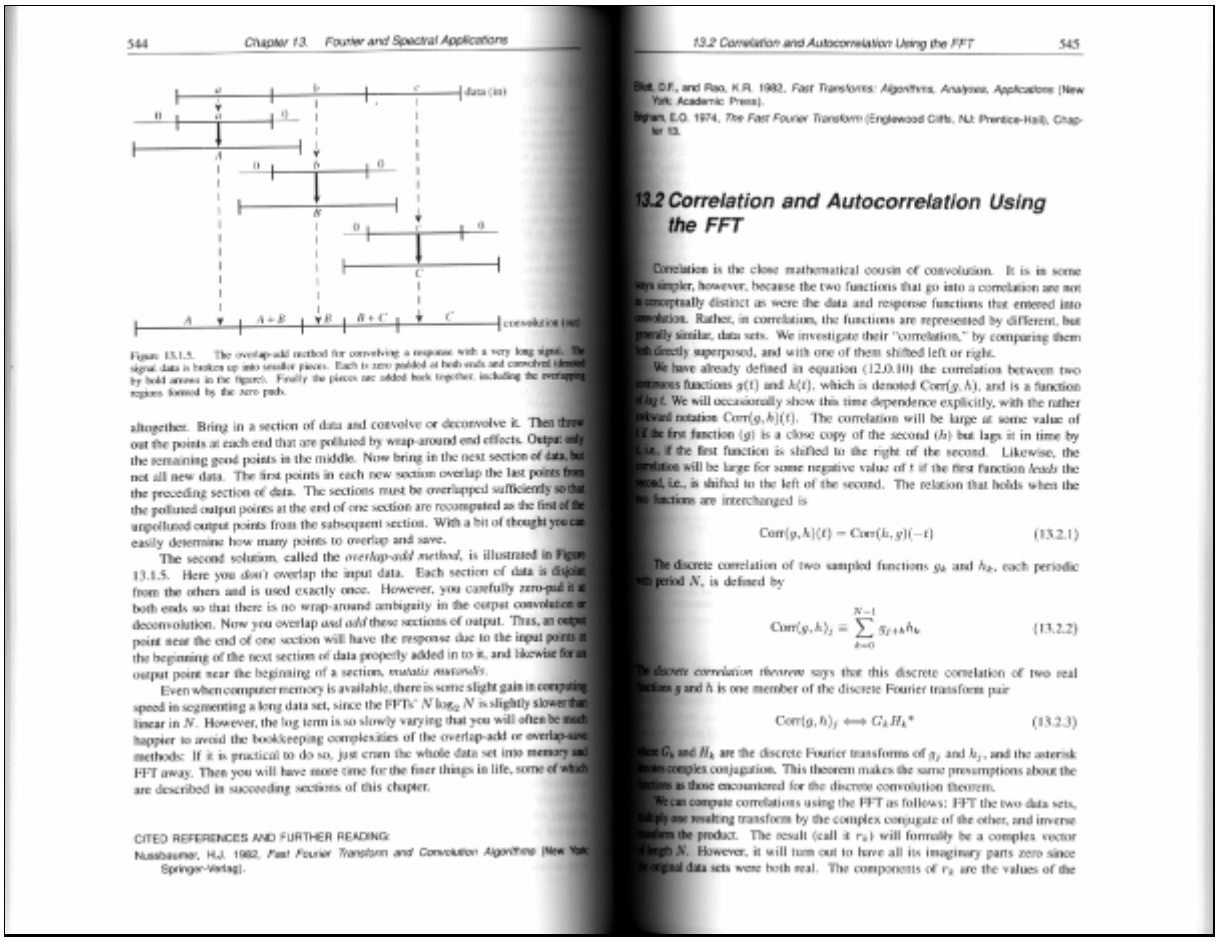


Figure 10. SuSE Linux 5.1 manual, 1997

Figure 11. *Running Linux*, Matt Welsh and Lars Kaufman, O'Reilly, 1994



544 Chapter 13. Fourier and Spectral Applications

Figure 13.1.5. The overlap-add method for convolving a response with a very long signal. The signal data is broken up into smaller pieces. Each is zero padded at both ends and convolved (denoted by bold arrows in the figure). Finally the pieces are added back together, including the overlapping regions formed by the zero pads.

altogether. Bring in a section of data and convolve or deconvolve it. Then throw out the points at each end that are polluted by wrap-around end effects. Output only the remaining good points in the middle. Now bring in the next section of data, but not all new data. The first points in each new section overlap the last points from the preceding section of data. The sections must be overlapped sufficiently so that the polluted output points at the end of one section are recomputed as the first of the unpolluted output points from the subsequent section. With a bit of thought you can easily determine how many points to overlap and save.

The second solution, called the *overlap-add method*, is illustrated in Figure 13.1.5. Here you *don't* overlap the input data. Each section of data is disjoint from the others and is used exactly once. However, you carefully zero-pad it at both ends so that there is no wrap-around ambiguity in the output convolution or deconvolution. Now you overlap *and add* these sections of output. Thus, an output point near the end of one section will have the response due to the input points at the beginning of the next section of data properly added in to it, and likewise for an output point near the beginning of a section, *mutatis mutandis*.

Even when computer memory is available, there is some slight gain in computing speed in segmenting a long data set, since the FFT's $N \log_2 N$ is slightly slower than linear in N. However, the log term is so slowly varying that you will often be much happier to avoid the bookkeeping complexities of the overlap-add or overlap-save methods. If it is practical to do so, just cram the whole data set into memory and FFT away. Then you will have more time for the finer things in life, some of which are described in succeeding sections of this chapter.

CITED REFERENCES AND FURTHER READING:

Nussbaumer, H.J. 1982, *Fast Fourier Transform and Convolution Algorithms* (New York: Springer-Verlag).

13.2 Correlation and Autocorrelation Using the FFT 545

Elliott, D.F., and Rao, K.R. 1982, *Fast Transforms: Algorithms, Analyses, Applications* (New York: Academic Press).

Brigham, E.O. 1974, *The Fast Fourier Transform* (Englewood Cliffs, NJ: Prentice-Hall), Chapter 13.

13.2 Correlation and Autocorrelation Using the FFT

Correlation is the close mathematical cousin of convolution. It is in some ways simpler, however, because the two functions that go into a correlation are not as conceptually distinct as were the data and response functions that entered into convolution. Rather, in correlation, the functions are represented by different, but generally similar, data sets. We investigate their "correlation," by comparing them both directly superposed, and with one of them shifted left or right.

We have already defined in equation (12.0.10) the correlation between two continuous functions $g(t)$ and $h(t)$, which is denoted $\mathrm{Corr}(g, h)$, and is a function of lag t. We will occasionally show this time dependence explicitly, with the rather awkward notation $\mathrm{Corr}(g, h)(t)$. The correlation will be large at some value of t if the first function (g) is a close copy of the second (h) but lags it in time by t, i.e., if the first function is shifted to the right of the second. Likewise, the correlation will be large for some negative value of t if the first function *leads* the second, i.e., is shifted to the left of the second. The relation that holds when the two functions are interchanged is

$$\mathrm{Corr}(g, h)(t) = \mathrm{Corr}(h, g)(-t) \tag{13.2.1}$$

The discrete correlation of two sampled functions g_k and h_k, each periodic with period N, is defined by

$$\mathrm{Corr}(g, h)_j \equiv \sum_{k=0}^{N-1} g_{j+k} h_k \tag{13.2.2}$$

The *discrete correlation theorem* says that this discrete correlation of two real functions g and h is one member of the discrete Fourier transform pair

$$\mathrm{Corr}(g, h)_j \Longleftrightarrow G_k H_k^* \tag{13.2.3}$$

where G_k and H_k are the discrete Fourier transforms of g_j and h_j, and the asterisk denotes complex conjugation. This theorem makes the same presumptions about the functions as those encountered for the discrete convolution theorem.

We can compute correlations using the FFT as follows: FFT the two data sets, multiply one resulting transform by the complex conjugate of the other, and inverse transform the product. The result (call it r_k) will formally be a complex vector of length N. However, it will turn out to have all its imaginary parts zero since the original data sets were both real. The components of r_k are the values of the

Figure 12. *Numerical Recipes in C,* Press, Teukolsky, Vetterling and Flannery, Cambridge University Press, 1992

Windows books are usually produced with FrameMaker and have a very similar look.

Math books

Of course I ought to include a couple of math books too, since MAPS is about TEX, and math typesetting is the first and foremost reason why TEX has been created.

'Numerical Recipes (figure 12) is indeed typeset with TEX. Running text is set in Times, headlines and references in Helvetica, math in Computer Modern.

As often with textbooks, the lines are a bit long for comfort. I would have liked smaller pages, but then there would have been even more of them, making the book even more unmanageable.

Equations are centered, and paragraphs are always indented, also after a section head or an equation. Jarring is the combination of section numbering and a title which doesn't fit on one line.

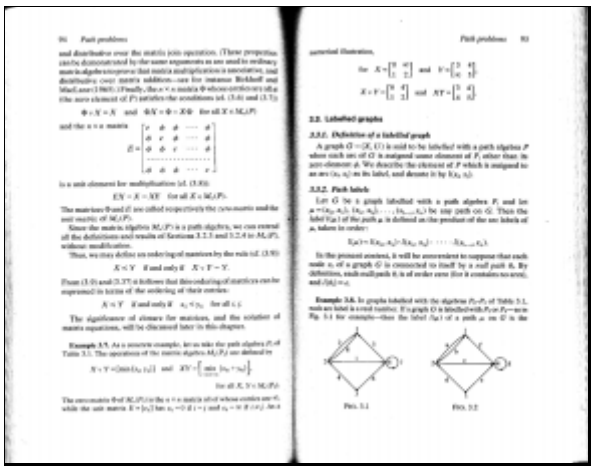


Figure 13. *Graphs and Networks,* Bernard Carré, Oxford University Press, 1979

Our second math example, Graphs and Networks (figure 13), must have been typeset without TEX. It also centers equations and indents all paragraphs. Here, the page size is comfortable. On the minus side, all the math and the numbered section heads and the illustrations make for a very busy layout.

I don't know what it would take to make such publications look good. Detailed and numbered sectioning often is a necessity; left-aligned equations would still look busy although maybe a little bit less so.

A picture book

We end with a cute little book on a grim subject: 'History of the war' (figure 14). It has the dimensions of an ordinary paperback, except for its landscape orientation. This alone already makes for a striking effect. It is printed in two colors, orange and black. Each spread is about a separate topic. It is designed on a three-column grid. In the displayed spread, only one of the two times three columns is filled with running text.

Justification is not very good, but it hardly matters because of the strength of the general layout and the impact of the illustrations.

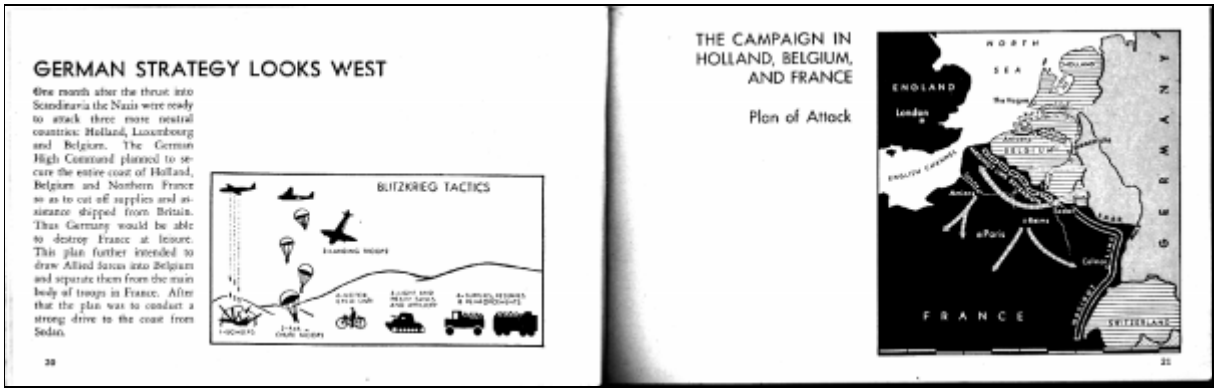


Figure 14. *A history of the war*, Rudolf Modley, Penguin, 1943

conference report

Report on TEX-Tagung Dante'99 in Dortmund

Erik Frambach, Taco Hoekwater, Siep Kroonenberg

abstract

A report on Dante's 20th meeting which took place in Dortmund. The future of the NTS project and the ε-TEX project were hot issues that were discussed during this meeting.

keywords

report, Dante, NTS, ε-TEX

Introduction

From February 24 to 26 the German TEX users group Dante held its 20th meeting in Dortmund. The NTG was represented at this meeting by Hans Hagen, Taco Hoekwater, Wietse Dol, Siep Kroonenberg and Erik Frambach. There were several reasons for this large delegation. This year Dante is celebrating its 10th birthday. The meeting was also an opportunity to get the NTS team and the ε-TEX team together to discuss the future of these projects.

Arriving in Dortmund

In Dortmund Dante had kindly arranged for rooms in a hotel. When we arrived there around 22:30 the hotel personnel told us that the rooms were no longer available because we should have arrived before 18:00. Naturally we complained about this because no such conditions were written on their confirmation fax that we showed them. So, the hotel personnel reluctantly gave us the rooms anyway. No excuses. The rooms were fine, though, and the breakfast excellent.

February 24

Registration started at 9:00 and there was a cute surprise for all participants. We all got a little TEX lion that can be attached to a shirt or a tie. After registering the meeting began. Dante's president Thomas Koch spoke a few words to welcome everyone, and then parallel tutorials on different subjects started. Below we will give an impression of the program.

Axel Reichert spoke about *Typographie – Gestaltung einer Beispielklasse*. He explained what the essential parameters are of a well-designed document from a typographer's point of view. Then he showed us which currently available LATEX packages can be used to implement these ideas. A nice overview.

David Kastrup spoke about *De Ore Leonis – Makroexpansion für Virtuosen*. His explanations of precisely how and when TEX input expansion actually takes place were so clear that we have asked him to provide a few pages for the next MAPS issue.

In the afternoon Thomas Feuerstock gave an *Einführung in PDFTEX*. He showed what extras PDFTEX offers and how it differs from 'normal' TEX e.g. with respect to graphics inclusion. In parallel Bernd Raichle and Thomas Koch presented *LATEX für Puristen – oder: Wie sauberes Markup dein Leben erleichtert*.

Heiko Overdiek elaborated on the *Gestalltung der Lesezeichen und Informationseinträge in PDF-Dateien mit dem Paket 'hyperref'*. No less than 70 sheets were presented. Some of the gory details were clearly too much for the audience, and unfortunately the lecture was based on a rather outdated version of pdfTEX. In parallel an *Einführung in LATEX 2ε* was given by Arnulf Liebing and Günter Partosch.

In the late afternoon there was an excursion to the *westfälischen Industriemuseum*, followed by a nice diner in town.

During the day and evening the NTS team and the ε-TEX team gathered to discuss and exchange information on the current state of affairs.

On the subject of ε-TEX, an agreement has been reached to discontinue the current ε-TEX team within the NTS working group. ε-TEX does not need funding in it's current state anyway, and the removal of formal ties with NTS gives Peter Breitenlohner (the 'new' head of ε-TEX) the possibility to concentrate on real programming issues. Peter announced that he has a desire to maintain having the current group of people connected to ε-TEX and these people all agreed to cooperate on extending ε-TEX, so the continuity of the project is still guaranteed.

The NTS project seems to be doing well, but it appears that there have been communication problems in the past. During the discussion, most of these problems were sorted out. NTS now consists of *only* the Java NTS group, thereby simplifying the organizational structure of the project considerably. It was agreed upon that an independant report should be written on NTS progress to clear the air. Our own Hans Hagen volunteered to do this.

Karel Skoupy presented a demonstration of the NTS work done so far: almost all of the mode-independent processing is now finished, as well as the reading of TFM files and writing of DVI files. In the next four months, the typesetting portions will be inserted into this framework. At EuroTEX '99 in Heidelberg, the NTS implementation of TEX'82 should be finished. After that has been done, Karel can start adding in some of the extensions that are now available in for instance ε-TEX and pdfTEX.

On the subject of pdf(e)TEX, Peter remarked that while pdfeTEX has his blessing as a valid implementation of ε-TEX, he himself will not spend time on developing it because of the fact that pdfTEX is completely non-portable web2c-only software. This implies that while the current code for pdfTEX is unusable by either ε-TEX or NTS, both groups *are* interested in the added functionality. This is a comforting thought for the future.

February 25

The day started with a word of welcome from the rector of the University of Dortmund, Prof.Dr.Dr. Klein. Next Prof.Dr. Reusch explained the history and organization of the university. Then Thomas Koch, president of Dante, welcomed us once more, and chaired the *Mitgliederversammlung*.

In the afternoon Gerhard Friesland-Köpke gave a lecture on *TEX82 – Rückblick und Wirkungen*. Georg Lachenmayr discussed *Scriptum ür eine technische Vorlesung – eine ganz normale Anwendung für LATEX?!*, while in parallel Werner Lemberg showed *TrueType-fonts mit TEX nutzen – das TTF2PK-Paket*.

Hans Hagen repeated the 'Hermann Zapf' experiment that was conducted previously at the NTG meeting in Leuven in 1998. This experiment tests the performance of an optimization of character widths. The optimization routine was changed since last time, which explains why the results were completely different. At the GUST Bachotek meeting in May a similar experiment will be conducted, and of course a report on the results will appear in MAPS.

Taco Hoekwater discussed *TEX at Kluwer Academic Publishers, Dordrecht. –Now and in the future–*, while in parallel Axel Reichert discussed the *Satz von Tabellen*.

Then the second part of the *Einführung in LATEX 2ε* by Arnulf Liebing and Günter Partosch was presented, while in parallel Klaus Höppner discussed *Die fpTEX-Distribution für Windows 9x/NT*.

The last lecture of the day was by Wolfgang Kühn who presented *iTe, der Interaktive TEX Editor*.

For the evening Dante had arranged tickets for the *Varieté Luna*, which was attended by many people. The foreigners (also from the Czech Republic and from Great Britain) decided to go into town instead to have diner and ponder over the future of documents and information systems.

February 26

The day started with Matthias Eckermann's lecture on *Von Kamelen und anderem Getier – Das „Kommentierte Vorlesungsverzeignis" der Katholisch-Theologischen Fakultät München*. In parallel Hans Hagen talked about *Context: what it is and how it works, an introduction*. As we have come to expect, his talk sent a wave of Ohh's and Ahh's through the audience.

Werner Lemberg introduced us to *Sprachen jenseits von Babel – das CJK-Paket*, while in parallel Taco Hoekwater talked about *Extending TEX for XML processing*. This talk presented eetex, which has it's own article elsewhere in this MAPS issue.

The last lecture of the meeting was again by Hans Hagen who gave us a feeling of 'Alice in Wonderland', as one of the participants expressed it: *Going beyond traditional paper documents*. Many, many examples showed us possible new way to handle documents on screen using technologies like pdfTEX and Javascript working together with TEX and MetaPost.

Thomas Koch then officially closed the meeting, thanking all the participants and the organizing committee for their excellent work.

Conclusion

The meeting was interesting and well organized. Surprising to us was the fact the Dante's 10th anniversary was mentioned only briefly. There was no celebration of any kind.

In general it's always interesting to see how our sibling user groups arrange meetings and manage their affairs. There are lots of things we can learn from each other. We hope further mutual visits will make the bonds between user groups even stronger. It's good to have friends everywhere; we expect to see some of our german friends attending the next NTG meeting.