Redactioneel

Taco Hoekwater
Siep Kroonenberg
maps@ntg.nl

Alweer een dunne en late MAPS. We hopen dat de kwaliteit dit een beetje goedmaakt.

We moeten nog eens ernstig nadenken over organisatie en procedures. Als twee mensen in dezelfde organisatie het werk doen dan kan er wat dat betreft niet veel misgaan; nu we proberen contacten met auteurs en proeflezen los te koppelen van produktie blijkt een strakkere organisatie noodzakelijk om te garanderen dat er daadwerkelijk iets gebeurt.

Bovendien hebben we het zonder een aantal vaste auteurs moeten stellen: ditmaal geen 'gezeefd', 'maar' negen pagina's van Hans Hagen, en van Taco zelfs helemaal niets. We zouden graag zeggen dat jullie de ontbrekende pagina's nog tegoed hebben voor het volgende nummer, maar we willen niet opnieuw beloftes doen die we misschien niet waar kunnen maken: we moeten wel iets hebben *om* af te drukken.

'De NTG en het internet' zijn we deze keer doelbewust overgeslagen. Het kwam met de katernen deze keer precies uit, er zijn namelijk in totaal 96 pagina's binnenwerk. Als we de internet pagina's er nog bij hadden gedaan hadden we net twee pagina's extra gehad, resulterend in een extra katern en daarmee extra drukkosten. Bovendien veranderen de gegevens over de NTG lijsten niet zo snel, één keer per jaar afdrukken is eigenlijk meer dan genoeg.

Wat staat er dan wel in deze MAPS, hoor ik je vragen? Wel, 'Praten met drukkers' laat zien waarom auteurs geen tijd hebben om voor de MAPS te schrijven. De geschetste problemen en misverstanden zijn om moedeloos van te worden, maar het is wel een leuk stukje geworden. De boze buitenwereld is nu eenmaal iets waar we allemaal mee om moeten gaan.

Verder hebben we een leuke verzameling toolbox-achtige stukjes deze keer. Van verschillende auteurs (Victor Eijkhout, Hans Hagen, Sven Bovin, Siep Kroonenberg en Erik Frambach) ontvingen we lekker korte artikeltjes met een hoop afwisseling. En natuurlijk weer de toolbox zelf.

Het literate programming artikel van Michael Guravage is een laatkomertje: dit artikel hoort bij de door hem gegeven lezing tijdens de vorige voorjaarsbijeenkomst. Een korte introductie in zowel de theorie als het gebruik van literate programming.

Thierry Bouche's artikel is alvast een vooruitblik op de volgende bijeenkomst: het zetten van poezie is zonder enige twijfel een voorbeeld van extreem gebruik van TEX.

Eindelijk horen we ook weer eens iets over Omega. Dit artikel is het enige artikel in deze MAPS dat 'traditioneel', dat wil zeggen via (o)dvips en de Adobe Distiller is gegenereerd. Er is de afgelopen paar jaar een hoop veranderd aan Omega, en het Omega team heeft beloofd dat we voor de volgende MAPS een artikel krijgen over alle nieuwigheden. Afwachten maar.

Als laatste een uitgebreid (hoewel nog niet volledig) artikel van de hand van Berend de Boer, met als doel voor een gemiddelde gebruiker de overstap van LATEX naar CONTEXT wat gemakkelijker te maken.

Colofon

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

De gebruikte fonts zijn Adobe Times-Roman+Old Style Figures en Frutiger, met een versmalde Courier voor de 'verbatim' omgevingen.

Compilatie gebeurt met web2c versie 7.3.1 onder Linux en Windows 95, en we drukken van een PDF bestand dat wordt gemaakt met pdftex 0.14 en dvips versie 5.86 (gedistilleerd met Adobe's Distiller 4.05), op 90-grams coated papier.

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

ntg
NTG- en TEX Info

Diverse afkortingen

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

AllTEX – TEX, LATEX, etcTEX

ltxiii – LATEX 3.0

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

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

AMS – American Mathematical Society

SGML – Standard Generalized Markup Language

TEX kalender

- 10–12 mei: GUTenberg 2000. Thema: LATEX et XML: coopération pour l'internet. Toulouse, Frankrijk.
- 15 juni: De 25ste bijeenkomst van de NTG. Het thema is TEXtremiteiten. Rijksuniversiteit Groningen.
- 12–18 augustus: TUG2000, de 21ste globale TUG conferentie. "TEX enters a new millennium". Wadham College, Oxford, Engeland.

NTG/TUG lidmaatschap

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

MAPS 00.2

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

1 juli 2000 (MAPS 00.2; #25)

15 januari 2001 (MAPS 01.1; #26)

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

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

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

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

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

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

de boze buitenwereld

Praten met drukkers

Een coproductie van de MAPS-redactie en het NTG-bestuur

Maarten Gelderman[1]

Praten met de VU-drukker

We schrijven januari 2000. Met typerende doortastendheid heeft de huisdrukkerij van de Vrije Universiteit de ultieme oplossing voor het millenniumprobleem bedacht. Per begin van het jaar worden twee zwart-wit- en een kleuren-DocuTech het hoofdgebouw van de VU binnengedragen.

Met de komst van de DocuTech's gaat de universiteit een nieuwe fase in. Syllabi en tentamens worden niet langer vanaf papier verwerkt, de voorkeur gaat nu uit naar digitaal aanleveren onder het motto: meer flexibiliteit en hogere kwaliteit. Dat laat ik met geen twee keer zeggen. Via het secretariaat laat ik informeren welke bestandsformaten onze vriend de drukker kan verwerken en of hij ook in staat is ZIP-schijfjes in te lezen. Er zit een syllabus van bijna 300 pagina's aan te komen en dat gaat nooit op een flop passen. Het antwoord is er binnen een dag: de drukker ontvangt bij voorkeur Word-bestanden en ze maken gebruik van WinZip.

Dat was niet helemaal mijn idee van bruikbare informatie. Een trip naar de catacomben van onze betonnen toren—ivoor zit er met het wetenschapsbudget van de heer Hermans niet in—leert me dat ze ZIP-schijven wel degelijk in kunnen lezen en bovendien weten dat ze PostScript kunnen verwerken.

Twee dagen later ligt de ZIP-schijf met de definitieve versie van mijn syllabus—het PostScript bestand is 55 MegaByte groot—bij de drukker. Twee weken later belt de afdelingssecretaresse mij thuis op: of ik onmiddellijk naar de VU kan komen, de drukker heeft een probleem. Een korte fietstocht naar de VU gevolgd door een bezoek aan de drukker leert me dat er *geen* 250 misdrukken het apparaat uit zijn gekomen. Gelukkig is het geheel ook niet per ongeluk enkelzijdig gedrukt: mijn onder Linux geformateerde ZIP-schijf (b)lijkt op hun drive onleesbaar. Inlezen op een andere PC, formateren en weer opslaan blijken de oplossing. Veiligheidshalve biedt ik spontaan aan om mee te kijken of alles nu wel goed gaat.

De drukker toont mij trots zijn DocuTech's, twee indrukwekkende scanners en een tweetal PC's waarmee hij geacht wordt het geheel aan te sturen. Terwijl de schijf in de op de parallelle poort aangesloten ZIP-drive verdwijnt wordt de digitaliseringsstrategie uitgebreid toegelicht en opgehemeld. 'En werkt het allemaal een beetje naar wens?' 'Nou weet u meneer, eigenlijk bent u de eerste.' Ik besluit niet weg te gaan voor ik weet dat die meneer mijn PostScript-bestand kan printen. De wetenschap dat het bestand een adequate PostScript-interpreter vereist (hij doet het prima met GhostScript en op mijn Lexmarkje van nog geen 1000 piek, peperdure HP-machines lusten er echter geen brood van) versterkt mijn wantrouwen.

Van enige aarzeling is echter voorlopig bij de drukker geen sprake. Onder het toeziend oog van een viertal collegæ die de geboorte van de eerste digitale syllabus ook wel mee willen maken opent hij één of andere vaag programma en gaat in het menu bestand–openen op zoek naar de ZIP-schijf. Binnen vijf minuten is dit ding gevonden. Uit een korte blik op de gebruikte bestandsextensies heb ik in de tussentijd afgeleid dat het gehanteerde programma dient voor het previewen van (uit de RIP afkomstige) bitmaps. De drukker kijkt mij in de tussentijd verwijtend aan en deelt me mede dat ik een lege schijf heb meegenomen. Ik heb kennelijk een goed humeur, want in plaats van de man de muis uit handen te drukken, leg ik hem goedgemutst uit dat mijn bestandsnaam op PS eindigt en hier dus niet te zien is. 'Ja, daar heeft die meneer die de cursus gaf ook wat over gezegd: dan moeten we naar de verkenner en het bestand kopiëren naar die machine daar.'

De verkenner wordt geopend, de ZIP-schijf dit keer sneller gevonden dan de vorige keer en tot mijn grote opluchting en bijna niet minder grote verbazing zie ik het vertrouwde gezicht van het GhostScript-spook verschijnen. Dat is goed nieuws: als de man het ding niet kan afdrukken kan ik hem in ieder geval laten zien dat mijn bestand in orde is. Het kopiëren van mijn bestand, dat uiteindelijk zeker 10 minuten blijkt te duren, kan beginnen. In de tussentijd krijg ik een lesje 'toekomstplannen van de huisdruk-

1. Alhoewel dit artikel in de eerste persoon enkelvoud geschreven is, is het zoals de titel al aangeeft in feite een coproductie van de MAPS-redactie en het NTG-bestuur. Waar er hier fouten staan, zijn die door mijn onwetendheid veroorzaakt. De slimme opmerkingen zijn afkomstig van Frans Goddijn, Hans Hagen, Taco Hoekwater en Siep Kroonenberg.

ker'. Het kennelijk ter bevordering van de klantentrouw door Rank Xerox ontwikkelde idee (waar de huisdrukker trouw mee wegloopt) is dat van digitaal aangeleverde documenten de geRIPte versie door de drukker bewaard wordt. 'Handig hè meneer, als u dan twee pagina's in uw syllabus wijzigt, hoeft u ons alleen maar een floppy met die pagina's te geven.' Ik geef het toe, voor een archief met scans van artikelen zie ik mogelijkheden, voor het overige getuigt dit verhaal van naïviteit en onwetendheid. Sinds een paar jaar weet ik gelukkig ook dat het verstandiger is de drukker mijn zienswijze niet op te leggen, dus al doorkeuvelend wachten we het verder kopiëren van het bestand af. De eerste collegæ van de drukker besluiten de bevalling van hun eerste digitale gedrocht niet verder af te wachten en te opteren voor de beschuit met muisjes na afloop. Nadat ik, voor zover mogelijk ('Nee meneer, een printer heeft een resolutie en die kan je instellen, een scanner niet.') alle technische details van de apparatuur heb doorgrond, begint het kijken naar het vrolijk knipperen van het oranje lampje van de ZIP-drive en het groene van de harde schijf.

Eindelijk is het bestand van de ZIP-schijf af. De drukker kondigt aan dat we nu geen computer gaan gebruiken, maar een soort printer en samen lopen we naar één van beide DocuTechs, waar we na enig zoeken het PostScript-bestand op de harde schijf van de zojuist door ons gebruikte computer kunnen traceren (zijn die dingen echt niet op afstand te besturen?). 'Als ie nu gaat tellen is het goed,' meldt de drukker zelfverzekerd. Tellen doet ie, inderdaad, maar rondom pagina 250 is het over. Hier staat een door een Adobe-product gegenereerde bitmap. Ik verdenk Adobe er nog steeds van dat hun belangrijkste doelstelling bij het ontwikkelen van software is, ervoor te zorgen dat deze concurrerende PostScript-interpreters doet vastlopen en, of het opzet mag zijn of niet, het door mij veronderstelde doel wordt bereikt. De RIP geeft het op. Zo denkt de drukker er overigens niet over. Hij negeert—heeft u beste lezer, zich er weleens over verbaasd hoeveel foutmeldingen je de gemiddelde computergebruiker voor kunt schotelen, zonder dat hij er ook maar één opmerkt—de mededelingen van zijn kostbare speeltje volledig en vertelt mij dat mijn bestand kwijt is, maar dat is niet erg, want hij belt gewoon die meneer van de cursus op en die vertelt wel weer hoe dat terug te vinden. Als ik nu gewoon zorg dat hij mij telefonisch kan bereiken, komt alles in orde.

Het zij toegegeven: alles kwam in orde. Mijn bestand is teruggevonden, die meneer van de cursus heeft kennelijk goede connecties met Rank Xerox, want binnen 24 uur—goede service—werd er 'iets aan een machine veranderd' en naar het schijnt is zelfs de syllabus af. Althans, de huisdrukker claimt dat hij in de boekhandel ligt, de VU-boekhandel heeft het ding nog niet gezien.[2]

Praten met bestuursleden

Na dit hele proces voelde ik me net Frans Goddijn. Volgens mij zijn er mensen lid van TEX-NL enkel en alleen om Frans zijn uit het contact met de grote-boze-buitenwereld resulterende literaire uitspattingen te bewonderen; drukkers vormen van die buitenwereld geen onbelangrijke, maar zeker wel een boze componenent. Kortom tijd om de overige ervaringen met deze booswichten eens aan een minder vluchtig medium dan de email toe te vertrouwen, oftewel een overzicht van misverstanden en echte problemen.

Over PostScript

In die grote-boze-buitenwereld heb je geen huisdrukkers, maar grote-boze-drukkers. Bij huisdrukkers lopen aardige mensen rond die zich realiseren dat ze 'ook maar' een huisdrukker zijn en niet alles kunnen weten. Bij grote-boze-drukkers wordt verondersteld dat de klant nog nooit van zijn leven een boek heeft gezien. Bij een huisdrukker kan je met een korte sprint langs de balie direct kijken wat er nu echt gebeurt. De drukkers waar Frans mee moet samenwerken worden benaderd via een uitgever, die een contactpersoon heeft, die weer spreekt met een account manager die op zijn beurt via de productiemanager met de persoon kan praten die daadwerkelijk aan de knoppen zit. Deze laatste persoon is de gelukkige bezitter van een MacIntoy en dan begint de ellende. Een MacIntosh-gebruiker heeft geen bestanden maar kastjes, documenten en allerhande soorten plaatjes (vrij vertaald is een MacIntosh-gebruiker een Windoze-gebruiker die zich omdat zijn computer het meestal doet nog nooit heeft afgevraagd hoe het ding werkt; van dit laatste geluk blijven Windowsgebruikers verschoond, of in de woorden van Taco: 'Alle drukkers in Nederland die niet met moderne bestanden overweg kunnen gebruiken MacIntoshes (waarmee ik nog geen oordeel wil vellen over het besturingssysteem, maar wel over de gemiddelde gebruiker daarvan).'). Hoe leg je een MacIntosh-gebruiker nu uit dat hij een PostScript-bestand naar de printer moet sturen? De best functionerende optie is tot nu toe de Taco-font-leugen. Bestanden kent de MacIntosh-bestuurder niet, lettertypes wel. Een klein beetje goedgelovige drukker valt uit te leggen dat je 'ook niet snapt hoe het werkt', maar dat er op de meegenomen schijf een lettertype staat dat gedownload moet worden naar de printer. Daar heeft de man een knopje voor en waarom er nu spontaan papier uit het apparaat begint te komen snapt de man evenmin, maar de klus is daarmee wel geklaard.

2. In de tussentijd is de syllabus ook in de boekhandel beland en afgezien van de bekende kwaliteitsproblemen met DocuTech-uitvoer voldoet het geheel aan de verwachtingen.

Dan moet het natuurlijk wel een eenvoudige klus zijn die direct uitgedraaid kan worden. Complexer werk (boeken) vereist het maken van inslagschema's het toevoegen van paskruizen en al dat soort zaken. Nu moeten we niet alleen kwaad naar de drukker kijken, maar ook de hand in eigen boezem steken. Dvips (waarmee in de TEX-wereld het merendeel van de PostScript bestanden wordt gegenereerd) maakt 'vreemde' PostScript bestanden aan en lang niet alle Prepress-apparatuur is hiervan gediend. Gebruik van commerciële programma's is een mogelijke oplossing. Wie dvips blijft gebruiken zal (met de optie `-j0`) in ieder geval moeten zorgen dat volledige fonts in het PostScript bestand worden opgenomen.[3] Ook is het vaak wenselijk om (handmatig of met een kort Perl-scriptje) de door dvips toegevoegde bounding box te verwijderen. Gebruik van ps2ps kan ook helpen bij het maken van 'gemakkelijker verwerkbare' PostScript. Het door Acrobat Distiller halen van het PostScript-bestand om dit vervolgens (met de *Adobe* en niet met de MicroSoft printerdriver) weer om te zetten in een nieuw PostScript-bestand is ook mogelijk, maar kan tot zichtbaar kwaliteitsverlies leiden.[4] Een laatste ook niet onaardige oplossing (wederom van de hand van Taco): 'laten printen door een repro-bedrijfje (die zijn in de regel veel moderner) op hoge resolutie, en dan camera-ready spul opsturen naar de drukker (een iets intelligentere drukker komt hier zelf op . . .).'

Over Portable Document Format

Portable Document Format (PDF). De naam belooft veel goeds. Het klinkt alsof je met zo'n bestand bij iedere drukker aan kunt komen en alles vervolgens vanzelf in orde komt. Helaas komt in de vorige zin het woord bestand weer voor, en hierboven is het reeds aan de orde geweest: lang niet alle drukkers weten wat een bestand is (ik zie maar even af van het feit dan PDF pas zo'n vijf jaar bestaat en dus nog niet door alle drukkers verwerkt kan worden). In mijn naïviteit meende ik dat de oplossing voor dit probleem toch voor de hand lag: je vertelt die drukker gewoon dat hij Acrobat Reader moet downloaden van de web-site van Adobe. Vanuit meer ervaring in drukkersland sprekende, meende Taco deze oplossing onmiddellijk de wereld uit te moeten helpen. Gesteld dat het de drukker lukt een bestand te downloaden (dat valt ten slotte niet geheel uit te sluiten), dan zit die drukker daarna met een HQX-*bestand*. De tijd die je nodig hebt om die drukker uit te leggen wat hij met dat bestand moet doen, kan je beter besteden aan het naar de printer sturen van een PostScript-bestand.

Nog gevaarlijker zijn drukkers die wel iets kunnen met een PDF-bestand. Zij hebben kennelijk een applicatie (bij voorbeeld QuarkXpress) waarmee een PDF-bestand geopend kan worden. Dit kan naar het schijnt dodelijke gevolgen hebben. De fontencoding dreigt in het honderd te lopen en dat heeft gevolgen die verdacht veel lijken op de standaardproblemen die wij Word-gebruikers altijd verwijten.

Overigens hebben we de hulp van dit soort pakketten niet nodig om onze TEX-uitvoer om zeep te helpen. We zijn hier zelfstandig ook heel goed toe in staat. In PDF-bestanden kan gebruikt worden van een zo goed als onbegrijpelijk verschijnsel dat bekend staat onder de naam subset (van een font). Subsets zelf zijn niet problematisch. De problemen ontstaan pas wanneer meerdere PDF-bestanden (b.v. artikelen) worden samengevoegd tot een groter bestand (in dit geval een tijdschrift). Bij het verwerken van de subsets in het samengevoegde bestand kunnen dan problemen optreden. Combineer dit met een eventueel niet gevonden letter die vervangen wordt door een Courier en je krijgt te maken met een eindresultaat waar een minder traditioneel ingesteld grafisch ontwerper (type Beowulf-gebruiker) zich niet voor zou schamen.

De combinatie PDF en fonts heeft overigens meer onhebbelijke eigenschappen. Sommige PDF-lezers maken gebruik van een optimalisatie die er voor zorgt dat de fonts in het PDF-bestand niet worden gebruikt als er op de harde schijf een font met dezelfde naam aanwezig is. Bij deze substitutie wordt echter niet gekeken of beide fonts dezelfde encodings hebben, zodat alle fi-ligaturen overwacht door æ kunnen worden vervangen, etc. Gegeven de overdaad aan encodings en hercoderingen van fonts waarop de TEX-wereld zo trots is, zijn rampen meer dan waarschijnlijk.

Een laatste probleem is, naar ik begrepen heb, het aantal lettertypes dat in een document gebruikt wordt. Gebruik van een te groot aantal lettertypes kan tot problemen leiden. Gegeven de beperkingen van TEX lijkt het me onwaarschijnlijk dat zich in de praktijk problemen voor zullen doen. Gebruikers van de opvolgers van TEX zijn echter gewaarschuwd.

Over Raster Image Processors

Hierboven was sprake van een printer. Met de printer (afdrukeenheid) hebben we bij de drukker echter weinig te maken. Veel belangrijker is de RIP, de raster image processor. Een PostScript-bestand bevat met name vectorgebaseerde pagina-beschrijvingen, hetzelfde geldt voor een PDF-bestand. Om deze bestanden af te kunnen drukken moeten zij eerst worden omgezet naar een voor de afdrukeenheid geschikte bitmap. In de bitmap staat precies weergegeven waar er wel en waar er geen inkt op papier

3. Houd er wel rekening mee dat de licentie van het font dit moet toestaan.

4. Sommige drukkers weten wat voor apparatuur ze gebruiken en soms is er voor deze apparatuur een PPD beschikbaar (PostScript printer definitie die door de Adobe PostScript Printerdriver gelezen kan worden om opties voor de uitvoer in te stellen). Soms kan de drukker je zelfs aan dit bestand helpen.

moet komen. Bij normale laserprinters is de RIP geïntegreerd in de printer. Drukkers hebben voor het RIPpen normaalgesproken een aparte computer (alhoewel ze het ding vermoedelijk niet als een computer beschouwen) die soms gedeeld wordt door meerdere afdrukeenheden. Bij problemen met de verwerking van een PostScript-bestand dat de prepress programmatuur heeft overleefd (of geen verdere verwerking bij de drukker heeft ondergaan) ligt de 'schuld' vrijwel altijd bij de RIP. De PostScript-interpreter kan fouten bevatten, het geheugen kan tekort schieten, de PostScript-versie kan verouderd zijn (en bij voorbeeld geen gecomprimeerde bestandsformaten ondersteunen) etc. Afgezien van echte PostScript-bugs in de RIP kan een oplossing vaak weer worden gevonden in het gebruiken van ps2ps om het PostScript-bestand om te zetten in eenvoudiger versie.[5]

Over resolutie

Tot nu toe hebben we het gehad over communicatie. Impliciet ging het over (het gebrek aan) compatibiliteit tussen mensen, computers en software. De gebruiker die met drukkers te maken krijgt, dient zich ook van een aantal andere zaken bewust te zijn. Inslagschema's en de looprichting van het papier zijn relevante zaken, waar ik helaas te weinig van af weet om me hier aan een uiteenzetting te wagen. Een ander aspect is verschil in resolutie. Hogere resolutie leidt niet alleen tot een verbetering van de kwaliteit van het eindresultaat, er treedt ook een aantal andere veranderingen op. Zowel spatiëring als de 'kleur' van grijstinten worden beïnvloed.

De spatiëring is het lastigste punt, maar tevens relatief gemakkelijk op te lossen en ook nog eens relatief onschuldig. PostScript-bestanden die geen bitmaps bevatten horen resolutie-onafhankelijk te zijn. PostScript-bestanden uit dvips zijn dit niet.[6] Bij het spatiëren van de letters wordt rekening gehouden met de afdrukresolutie. Dit leidt, zeker op lagere resoluties, tot een zichtbaar mooiere afdruk.[7] Dit vereist echter wel dat we de resolutie van de afdrukeenheid kennen. Gebruik van een 600-dpi PostScript-bestand op een 1520-dpi afdrukeenheid leidt tot een lagere kwaliteit dan mogelijk is. Genereer het bestand op de juiste resolutie (die de drukker hopelijk weet en anders wellicht uit een PPD file valt af te lezen) en maak (met de optie `-D dpi`) de proefafdrukken op de verkeerde resolutie en niet andersom.

Voor grijstinten geldt dat de kleur van 10% grijs sterk afhangt van de resolutie en ook nog eens van de afdrukeenheid en het afdrukmedium. De kleur van een grijstint op een laserprinter vormt slechts zelden een goede benadering van de kleur die dezelfde grijstint heeft bij een betere afdrukkwaliteit. Op een 300-dpi printer is 10% grijs zo donker dat het nog net aanvaardbaar is als achtergrondkleur. Op 1200-dpi is de grijstint nauwelijks zichtbaar. Het handigst is het te werken met een proefprint waarop de 'kleur' van de verschillende grijstinten goed zichtbaar is. De meeste drukkers moeten deze print wel kunnen leveren. Mocht het maken van een verantwoorde inschatting van de kleur van een grijstint in het geheel niet mogelijk zijn, dan kan het werken met kleuren in plaats van grijstinten nog een optie zijn. De kleur 'grijs' die een geel vlak heeft lijkt minder variatie te vertonen dan de kleur van 20% grijs.

5. De grootte van de door ps2ps geproduceerde bestanden kan overigens extreem zijn, evenals het geheugengebruik van ps2ps zelf. Een test met het PostScript bestand van de eerder genoemde syllabus (oorspronkelijk 55 MB groot) resulteerde in een PostScript-bestand van maar liefst 273 MB voor de eerst 243 pagina's (daarna was mijn harde schijf vol, de uiteindelijke omvang van het vereenvoudigde bestand kan ik dus niet vaststellen).

6. Taco voegt hier aan toe: pas ook op bij gebruik van de Distiller, deze staan vaak ingesteld voor een beeldschermresolutie van 144-dpi, hetgeen ook niet tot optimale resultaten leidt.

7. In omgekeerde zin is dit vrij goed zichtbaar wanneer een dvips-PostScript bestand via psnup verkleind wordt afgedrukt. De resolutie klopt niet meer en de afdruk ziet er slechter uit dan nodig is.

being a wise guy

LaTeX met één toets vanuit vi

Sven A. Bovin
K.U.Leuven
departement Scheikunde
afdeling Kwantumchemie
Celestijnenlaan 200F
B-3001 Heverlee

abstract
De ontstaansgeschiedenis van een shellscript om vanuit vi met één toets LaTeX op te roepen en, indien nodig, xdvi te starten voor het previewen.

keywords
shell script, vi, LaTeX, Unix

Het begon allemaal met een tip op de `tex-nl`-discussielijst. Deze tip werd gepost door Hendri Hondorp en was afkomstig van de faculteit Informatica van de Universiteit Twente. Het is een manier om vanuit vi met één toetsaanslag LaTeX te laten lopen op je bestand en tegelijk je xdvi venster te laten verversen. Hiertoe wordt een executable script `do_latex` aangemaakt met volgende inhoud:

```
#!/bin/sh
echo ==========
echo q | latex $1 | tail -10
kill -USR1 `ps -a | grep xdvi | awk '{print $1}'`
```

Het script voert LaTeX uit en laat toont de laatste 10 regels van de output naar `std out`; indien nodig wordt LaTeX met `q` in batch-modus gezet na de eerste foutmelding. Om dit script uit te voeren vanuit vi, wordt een key mapping voorgesteld (in de .vimrc- of .virc-file): `map l :w^M:!do_latex %^M`. Hierin staat `^M` voor ctrl-M, in vi in te geven als Ctrl-V, Ctrl-M. Hierdoor krijgt de toetsaanslag `l` de betekenis 'sla het bestand op (w) en voer het externe commando `do_latex` uit (het scriptje) met als parameter de naam van het huidige bestand'.

Wybo Dekker merkte op dat de toets pijl-rechts bij hem ook een `l` genereert. Dit hangt samen met het default gedrag van vi, waarin naar rechts bewogen wordt m.b.v. de toets `l`. Hij stelt daarom voor een andere toest te mappen. Wybo vraagt bovendien of het mogelijk is xdvi te starten indien het programma nog niet loopt.

Johan Wevers antwoordt hierop:

```
!#/bin/sh
echo ==========
test `ps -ax | grep xdvi ` || xdvi $1
```

maar helaas is het niet zo simpel. `ps -ax` vangt ook xdvi's van andere gebruikers, en er moet bovendien gefilterd worden om het juiste proces te pakken te krijgen. Wybo stelt nu het volgende script voor:

```
!#/bin/sh
echo ==========
file=${1%.tex}
echo q | latex $file | tail -10
pid=`ps x | grep xdvi | sed -e '/grep/d; s/^ *\([0-9][0-9]*\).*/\1/;q'`
if [ x$pid == x ]; then
  xdvi $file&
else
  kill -USR1 $pid
fi
```

`file=${1%.tex}$` slaat de eerste buffer van vi, dankzij de key mapping is dat de naam van het huidige bestand, op in `file`, `s/^ *\([0-9][0-9]*\).*/\1/` isoleert het pid en de `q` in dezelfde regel zorgt ervoor dat je maar één xdvi ziet.

Bert Frederiks kan het nog mooier, en stelt

```
#!/bin/sh
echo ==========
file=${1%.tex}
echo q | latex $file | tail -10
pid=`ps x | grep " [x]dvi *$file" | awk '{print $1}'`
if [ -z "$pid" ]
then
  xdvi $file&
else
  kill -USR1 $pid
fi
```

Waarmee Wybo het gedeeltelijk eens is. Hij vindt echter

```
#!/bin/sh
echo ==========
file=${1%.tex}
echo q | latex $file | tail -10
pid=`ps x | grep " [x]dvi $file"`
if [ -z "$pid" ]
then
  xdvi $file &
else
  kill -USR1 ${pid:0:5}
fi
```

leuker.

installation and configuration

Building a TEX installation for distribution

Siep Kroonenberg
Kluwer Academic Publishers
Dordrecht
email siepo@cybercomm.nl

keywords
Texmf tree, Windows, Linux, Perl, Configuration

At Kluwer Academic Publishers, KAP in short, we use TEX for typesetting journals. Since there are obvious advantages to using a standardized distribution, we provide our typesetters with one on CD. This way we can be sure that the right fonts and stylefiles are available, and it also eliminates much of the guesswork when we need to investigate technical problems.

The installation is based on a Web2c TEX implementation. The binaries are from fpTEX [1], Fabrice Popineau's Web2c TEX implementation for Win32 platforms. There are two texmf trees; the main tree is based on a recent texmf tree from Thomas Esser's teTeX [2], and the 'local' tree contains some additions from various sources: the Kluwer stylefiles, some commercial fonts and some additional packages. I maintain a third tree for personal and internal use. In this tree I put NTG stuff and development versions of the Kluwer styles.

Maintenance and backups

The main argument for using multiple trees is simpler maintenance: updating major standard packages requires downloading a new texmf tree from Thomas Esser's site and applying a small number of fixes. Our own customizations reside in another tree and remain unaffected.

The combined size of the full zipped distribution is roughly 50MB. By far the largest chunk is the main texmf tree, which can easily be recreated from online sources and therefore doesn't require backing up. The remaining pieces are small enough to keep some old versions around: more for my own scripts, less for collections of downloaded material. As an additional safeguard I keep an informal log of my work, in which I write down what I got from where, among other things.

Linux and Windows

I use the TEX installation both under Linux and under versions of Windows. I'll explain below how I let both platforms share configuration files. Sharing format files is possible only if the binaries of both platforms have been built from matching sources. At the moment, this precludes the use of the latest versions of the binaries.

I much prefer Linux, but some things have to be done under Windows. Moving back and forth between Windows and Linux is a real hassle; because of the differences in file systems and textfile formats, textfiles may acquire DOS line endings, files may get their executable flags set, or capitalization of filenames may change. This is not only untidy but can also cause real trouble. The zip and unzip programs have some options to fix capitalization and line endings, but this is only a partial remedy.

A much more convenient solution is to maintain the TEX installation on a Linux system which runs Samba to make TEX available to Windows. If VMware [3] is installed, the Windows system may even be a virtual machine running under Linux on the same physical machine. But I haven't yet persuaded my boss to provide me with these goodies. See also the screenshot at the end.

TEXMF.CNF: cascaded configuration

For those readers who haven't delved into the mysteries of Web2c configuration, here a quick summary:

Most configuration of a Web2c TEX system is done in a file `texmf.cnf`, which is consulted by all Web2c components. There is a default location for `texmf.cnf` *relative to the location of the executables*, so that a standard Web2c installation will run 'out of the box' without any manual configuration. These are a couple of settings from the 'main' `texmf.cnf`:

```
TEXMFMAIN = $SELFAUTOPARENT/share/texmf
TEXMF = !!$TEXMFMAIN
```

The first line defines the location of the main texmf tree relative to the TEX executables, the second line specifies that this tree is the only tree (the exclamation marks mean that no searching outside the ls-R file database should be done).

But it is also possible to specify the location of `texmf.cnf` in an environment variable, or even to specify several directories. The `texmf.cnf` files in all the specified directories will be read, earlier settings taking precedence over later ones. So you can leave the original `texmf.cnf` untouched, and create a new higher-precedence one containing merely the settings which need to be changed. I

created texmf.cnf files for the local tree and for the private tree, so that the private tree can be added to my TEX environment merely by changing the TEXMFCNF environment varable.

The scheme is even more flexible because environment settings can also be used for the variables in texmf.cnf. If *e.g.* under DOS/Windows the following environment variables have been set:

```
SET TEXMFCNF=c:/tex/local/texmf/web2c;
  c:/tex/share/texmf/web2c
SET TEXMFLOCAL=c:/tex/local/texmf
SET TEXMFMAIN=c:/tex/share/texmf
```

and in our local texmf.cnf we have a line

```
TEXMF = {!!$TEXMFLOCAL;!!$TEXMFMAIN}
```

then both the local and the main tree will be active, the local tree having precedence. This same configuration file can be shared between Windows and Unix since the operating system-dependent names of the root directories are defined outside texmf.cnf. Note that, in texmf.cnf, semicolons for path separators and forward slashes for directory separators are acceptable to both DOS/Windows and Unix. The texmf.cnf file of the third tree defines

```
TEXMF = {!!$KLUDIR,!!$TEXMFLOCAL,!!$TEXMFMAIN}
```

where KLUDIR has been defined somewhere else.

A Perl installation script

An installation script involves some user- and some system interaction, and Perl is an obvious choice for the job.

Which Perl. I didn't want to assume or require that Perl be installed, let alone a specific version of Perl. So I included on the installation disk a copy of the executables and modules of the old ActiveWare Win32 implementation of Perl 5.003: it is good enough; it is much smaller than the current Perl [4]; it runs fine without explicit installation, and in the licensing conditions this use of this Perl is explicitly allowed without permission.

A couple of tricks:

Wrapping a Perl script in a batchfile. This is very old hat but I still think it is a neat trick. Perl has a command-line option -x which instructs Perl to skip to the first line starting with #! (shebang) and containing the word perl. A batchfile install.bat

```
.\perl -x install.bat
goto endofperl
#!perl
<perl code>
__END__
:endofperl
```

is first read by the command interpreter, which calls Perl with itself as input file, and then jumps to :endofperl, skipping over any Perl code. In its turn, Perl skips to its own code because of the -x option, and stops reading at __END__ stepping over any non-Perl code.

The installation script calls Perl as .\perl to make sure that the right copy of perl.exe gets called. In the actual script there is also a bit more code for the command interpreter than just the call to Perl.

A fake backquote. The Perl used for the installation has a backquote function: the output of a 'back-quoted' command can be assigned to a variable, e.g.

```
$pwd = `cd`
```

Unfortunately, it doesn't work under the original release of Windows 95. The following code fakes it (note that there were also some problems with redirection which had to be gotten around):

```
open TEMPBAT, ">$tempbat" ||
  die "can\'t make tempbat";
print TEMPBAT ($command, " >", $tempout, "\n");
close TEMPBAT;
system ($tempbat);
open TEMPFILE, "<$tempout" ||
  die "can\'t open tempfile";
my $sysout = <TEMPFILE>;
chomp $sysout;
close TEMPFILE;
```

Dialogs. At an earlier stage, I considered using Perl/Tk but dismissed this – with some regret – because Perl/Tk requires a newer Perl and the presence of DCOM, and that would make the installation procedure a lot more invasive. Instead, I just read user responses using <STDIN>. This doesn't allow for fancy user interaction, but will do. I have been told that modules exist, also under Win32, for more sophisticated character-mode dialogs.

Additional issues

Ghostscript. It seems unwise to disregard previously installed copies of Ghostscript. A more or less foolproof method to detect a preexisting version would require searching all local and network drives. In some cases this might be very slow. Instead, I check the registry and the environment (GS_LIB and the search path) and look in some obvious places.

Setting environment variables. I can think of some good arguments for setting environment variables globally

– for Windows 9x that would mean in `autoexec.bat`, but I opted for local settings, *i.e.* for a specific 'DOS box' or console, because this makes it a little bit simpler to have multiple TEX installations on one system, and it also saves some rebooting.

Shortcuts. Creating shortcuts is a bit of a problem. Their format apparently isn't documented. The proper way to create them is with Windows API calls, and the current Perl (not the one which is used for installation) even provides a function for it, but in the end I decided that doing it properly would take me too much time. Instead, I place prefab shortcuts on the desktop. If TEX was installed in a non-default location, then the user will have to manually change its path entries.

The right balance. On a less technical note: there is the matter of how far to go with user-friendliness and error handling. Time and again I am reminded that what is obvious or trivial to me isn't obvious or trivial to everybody. Although a really foolproof installation procedure is not an option, I try to keep in mind that a little bit more care on my side may save others considerable frustration and may help them decide to adopt the new distribution. On the other hand: the more safeguards are added, the harder it becomes to debug, test, maintain or modify the installation script. I am by no means sure that my scripts are anywhere near the optimum.

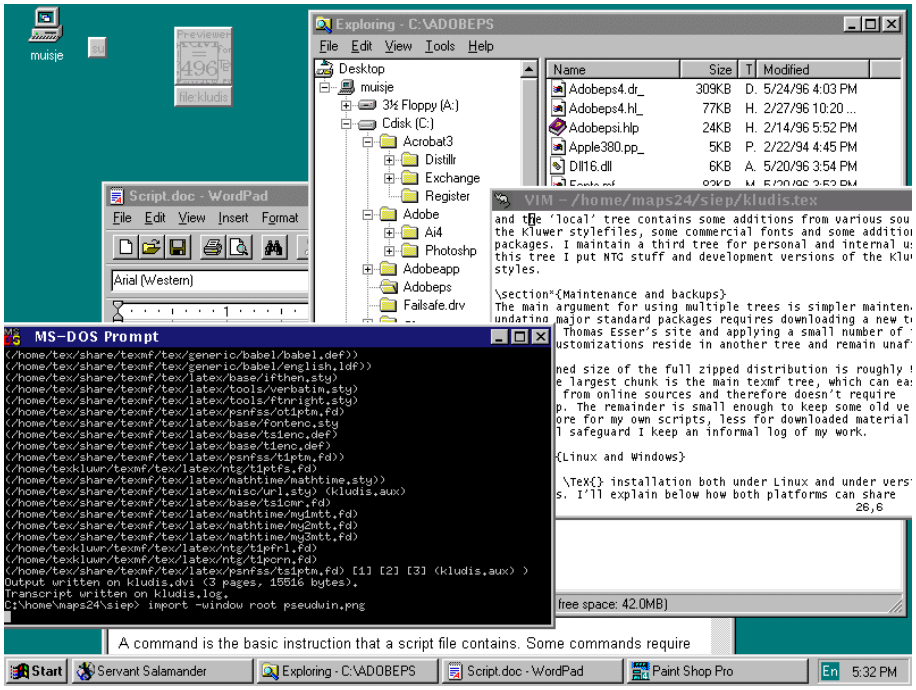
URL's

[1] The latest version of Fabrice Popineau's fpTeX can be downloaded from `ftp://ftp.ese-metz.fr/pub/tex/`.

[2] The latest teTeX can be downloaded from `ftp://ftp.rrzn.uni-hannover.de/pub/local/misc/teTeX-beta/`. There are separate archives for the sources of the executables and for the texmf tree.

[3] See `www.vmware.com`.

[4] The website of ActivePerl is `www.activestate.com`. I could no longer find any traces there of the old Actveware Perl.



Yet another way to combine Windows and Linux. This is actually Linux, with a Windows screenshot as wallpaper. The 'MS-DOS Prompt' is an rxvt X-terminal, with a tricked prompt command: `PROMPT_COMMAND='PS1='echo "C:$PWD> "|sed -e s,/,\\\\\\\\\\\\\\\\\,g'',` with thanks to Gilbert van den Dobbelsteen, Taco Hoekwater and Wybo Dekker.

Thierry Bouche

unexpected applications

Typesetting modern & contemporary poetry with LaTeX

abstract
TeX: a typesetting engine limited to scientific publishing? Where would be the fun?

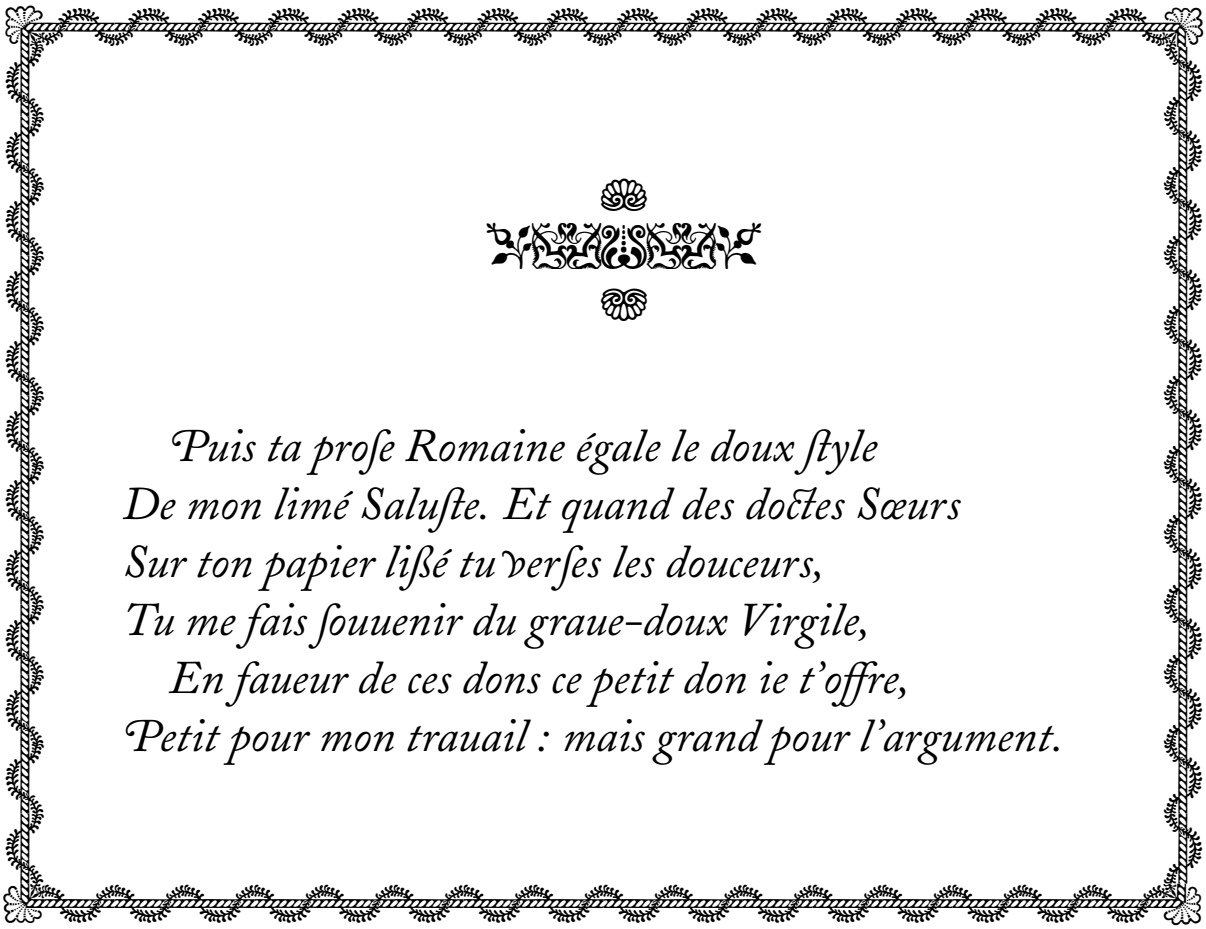
I

I learned TeX because I was a working mathematician. But 'science' never was my main interest field. At one point, I realized that TeX could do much more than the average desktop publishing software, when it comes to fine horizontal microtypography (vertical justification, grid fitting… being a quite weak point). My first experiment was with intelligent ligatures in virtual fonts, that allow to automate Renaissance use of the long *s* vs. the final *s*, or swash initials, through the word-boundary pseudo-charachter. I was also amazed how the border's ornaments were easily programmed using LaTeX's picture environment (figure 1):

```
\newlength\ornamentlength
\settowidth\ornamentlength{\fontsize{16pt}{16pt}\ornaments
qwwwwwwwwwwwwwwwwe}%
\begin{picture}(0,0)
\put(-3.5,-7.5){%
\parbox{\ornamentlength}{\fontsize{16pt}{16pt}\ornaments
qwwwwwwwwwwwwwwwwe\\[-3.5pt]
U\hfill u\\[-1pt]
  ............
U\hfill u\\[-1pt]
QWWWWWWWWWWWWWWWWE}%
}%
\end{picture}%

Puis ta prose  Romaine égale le doux style \\
De mon limé Saluste. Et quand des doctes S\oe urs\\
Sur ton papier lissé tu verses les douceurs,\\
Tu me fais souuenir du graue-doux Virgile,
```

figure 1: a contemporary version of a 1610 poem ☞



Puis ta proſe Romaine égale le doux ſtyle
De mon limé Saluſte. Et quand des doctes Sœurs
Sur ton papier liſſé tu verſes les douceurs,
Tu me fais ſouuenir du graue-doux Virgile,
En faueur de ces dons ce petit don ie t'offre,
Petit pour mon trauail : mais grand pour l'argument.

II

I was so happy with this Caslon experiment, that I used the automatic ligatures from the alternate fonts in a book I published some times later: this fitted quite the precious & even somewhat mannered style of J.-P. Bobillot. He was indeed very happy with the result!

The input source here has no special feature: I simply made a composite virtual font from the basic Adobe Caslon, its expert and alternate sets.

figure 2: A page of Bobillot's *Poèmes coupés* ☞

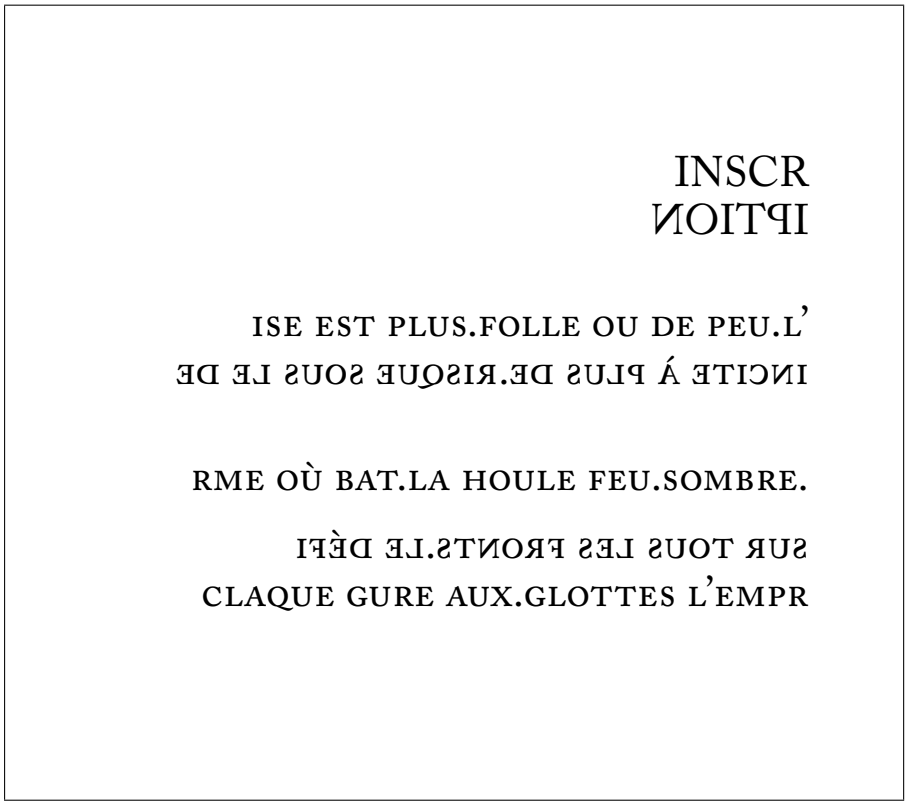
Poèmes coupés *est la réétricure & le présent volume contistue la version dénifivite de, respectrovitement :* La playmate des singes / psaume des paumes, *régidée en léthé 1985 & dont un état minitial fut publié, dans la collection « Plis », à Lompret, en 1986 ;* Autogënes slapsticks – intermezzo, *dérigés au prime-temps 1976 & dont un immature état fut oùblié, aux séditions « Le jeu des tombes », à Montmorency, également en 1986 ;* Promenade interdite (poëme), *dégirée à l'hot town 1986 & dont un format inimal fut publié, dans la collection « Tuyau (quotidien) », à Boulazac, en 1987. Il m'a toujours semblé que ces bribes débridées, caillasses de Petit Indexet rescapé des affres du Mot de l'Ordre, participaient d'un même* expéri/mentalisme lyrique. *D'où le* précipité *que j'ai tâché d'obtenir, ici, en les « télescopant ». Opération, réalisée les 27 & 28 novembre 1995, suivant le principe de la plus grande* hâte créatrice *: 20 ans « relus & colligés » en 2 jours !*

VOTRE TEMPS EST BREF, SOYEZ PRÉCIS !

Lyon, le 29 novembre 1995

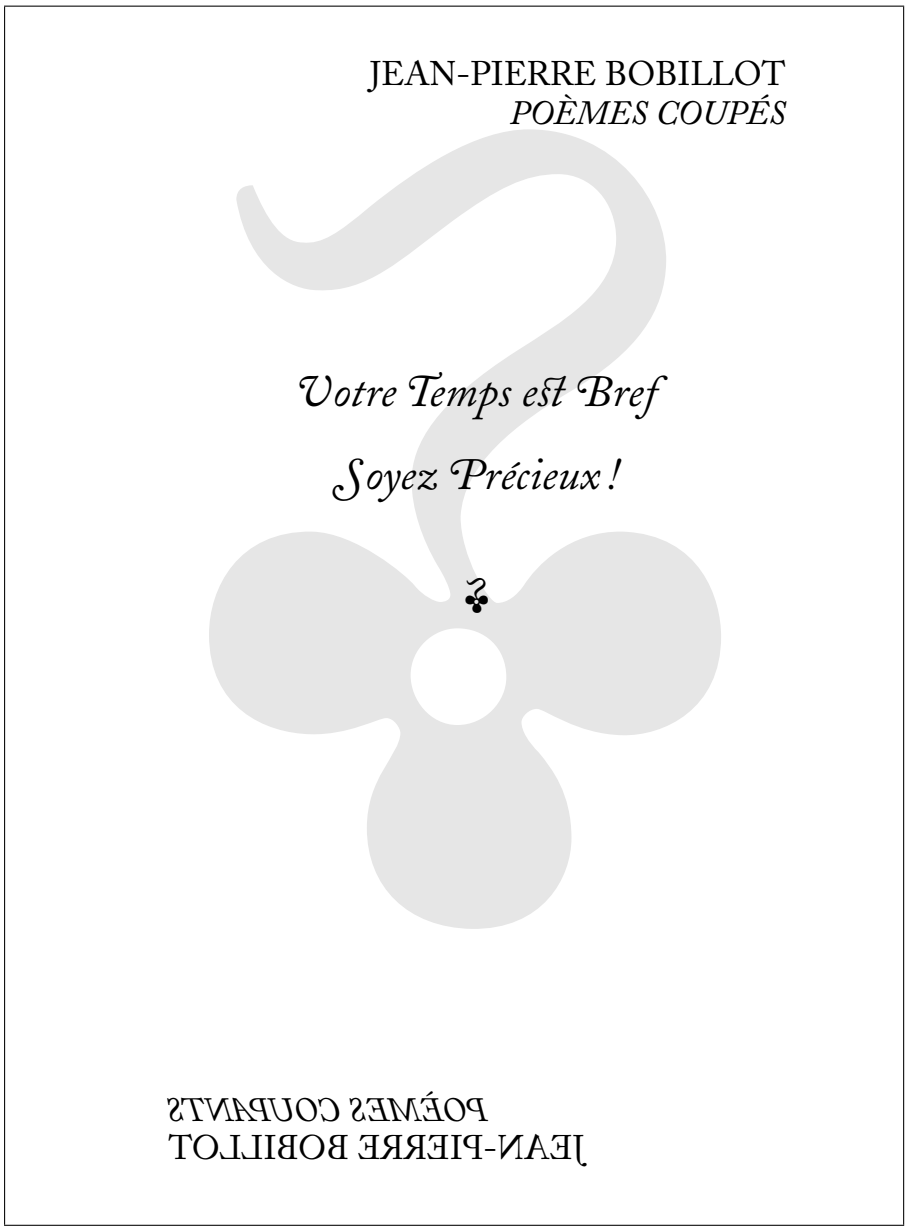
The same author had painfully written a ‘boustrophedon’ poem by inverting the letters: I proposed him to do it the right way, simply changing the direction of the typesetting at each newline (using *graphics*’ `\reflectbox` macro), which yields fig. 3.

figure 3: A modern boustrophedon! ☞


INSCR
IPTION
ISE EST PLUS.FOLLE OU DE PEU.L'
INCITE Ý PLUS DE.RISQUE SOUS LE DE
RME OÙ BAT.LA HOULE FEU.SOMBRE.
SUR TOUS LES FRONTS.LE DÉFI
CLAQUE GURE AUX.GLOTTES L'EMPR

Finally, I mixed both experiments in the fourth page of the booklet (this one is in Adobe Garamond—fig. 4).

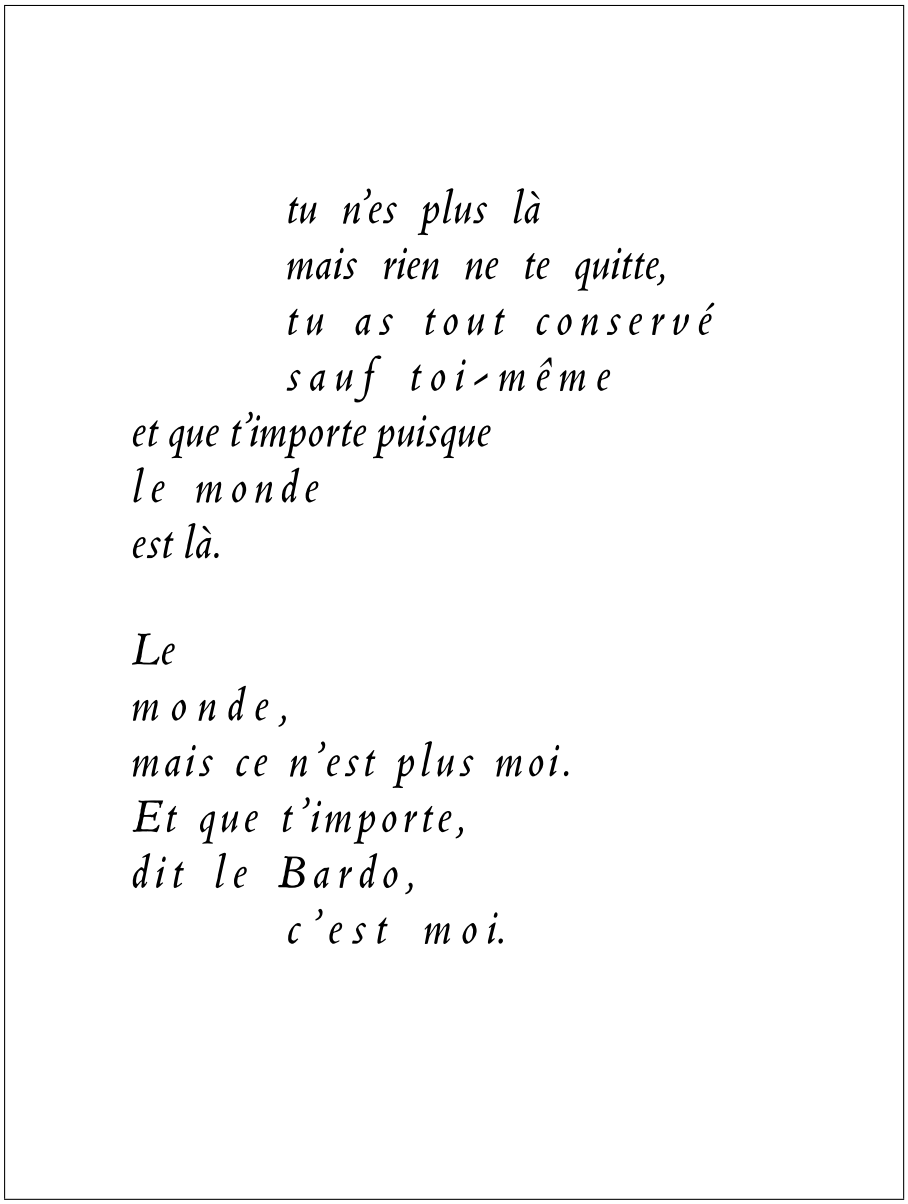
figure 4: Inverted sentence, inverted words. . . ☞


JEAN-PIERRE BOBILLOT
POÈMES COUPÉS
Votre Temps est Bref
Soyez Précieux !
POÈMES COUPÉS
JEAN-PIERRE BOBILLOT

III

Letterspacing is a strange beast, we're usually shocked by the horrors it may produce, when used blindly in some magazine's narrow columns. However, Antonin Artaud used it deliberately to induce reader's unrest. Here (fig. 5), I decided to render this discomfort by modifying the amount of letterspace on each line, and using the gorgeous Monotype Blado font, which I call a 'litteral humane', because it is one of the very seldom digital font that reproduce crudely a Venitian font *as it rendered in print* at that time.

figure 5: Inferno, according to Artaud ☞


tu n'es plus là
mais rien ne te quitte,
tu as tout conservé
sauf toi-même
et que t'importe puisque
le monde
est là.
Le
monde,
mais ce n'est plus moi.
Et que t'importe,
dit le Bardo,
c'est moi.

IV *Un coup de dés*

One of my hardest challenges was to typeset the seminal poem of Stéphane Mallarmé *Un coup de dés* with contemporary means, yet more faithfully to the author's intentions (as illustrated by some proof sheets he corrected, now kept in the Bibliothèque de France) than any version currently available in book stores. I tried out virtually any available digital didone before choosing Linotype Didot, which comes with a set of ornaments and special titling or initial fonts, and is very faithful to the most absolute Didot designs. In my opinion, the rigourous game that Mallarmé plays with sizes, relative weights, (non-) alignments, could not have been achieved with any sloppier font. Here, there is no special font trick (except that I had to convince a typographer friend to draw the missing ff ligatures for me), but I had to face the looseness of TEX's vertical typesetting. Getting rigourous horizontal alignments required to inhibit any glue and to make any vertical skip an integer multiple of `\baselineskip`. Apart from that, the work was relatively smooth, and TEX's quality is the first responsible for the beauty of the pages.

Title page ☞

Un coup de dés jamais n'abolira le hasard

poëme,
1897

COMME SI

Une insinuation

au silence

dans quelque proche

voltige

simple

enroulée avec ironie
ou
le mystère
précipité
hurlé

tourbillon d'hilarité et d'horreur

autour du gouffre
sans la joncher
ni fuir

et en berce le vierge indice

COMME SI

C'ÉTAIT
issu stellaire

CE SERAIT
pire
non
davantage ni moins
indifféremment mais autant

LE NOMBRE

EXISTÂT-IL
autrement qu'hallucination éparse d'agonie

COMMENÇÂT-IL ET CESSÂT-IL
sourdant que nié et clos quand apparu
enfin
par quelque profusion répandue en rareté
SE CHIFFRÂT-IL

évidence de la somme pour peu qu'une
ILLUMINÂT-IL

LE HASARD

Choit
la plume
rythmique suspens du sinistre
s'ensevelir
aux écumes originelles
naguères d'où sursauta son délire jusqu'à une cime
flétrie
par la neuralité identique du gouffre

V *Conclusion?*

The fine ‘best fit’ algorithm of TEX makes it a program of choice for the layout of any typeset material. In fact, it is the only tool I know of that allows to use the full features of extended font sets with lots of variants and contextual glyphs. Its scientific abilities even allow to put anything anywhere on the page, which may prove quite helpfull when dealing with contemporary poetry! Horizontal micro-typography is essentially perfect, as long as the fonts are well tuned, but the lack of control over vertical placement is somewhat tedious.

My next programmed experiment is to use pdfTEX’s HZ-like feature in a non-reasonable manner on a text by a french fool.

The Design and Use of a Multiple-Alphabet Font with Ω

Yannis Haralambous
Atelier Fluxus Virus,
187, rue Nationale, 59800
Lille, France.
Email:
yannis@fluxus-virus.com

John Plaice
School of Computer
Science and Engineering,
The University of New
South Wales, Sydney 2052,
Australia.
Email:
plaice@cse.unsw.edu.au

abstract
The Ω project aims to offer open and flexible means for typesetting different scripts. By working at several different levels, it is possible to offer natural support for different languages and scripts, and strictly respect typographical traditions for each of them. This is illustrated with a large PostScript Type 1 font for the commonly used left-to-right non-cursive alphabets, called `omlgc` (Ω Latin-Greek-Cyrillic). This font, which more than covers the Unicode sections pertaining to those alphabets, as well as those of IPA, Armenian, Georgian and Tifinagh (Berber), is built—virtually—out of smaller glyph banks. The Ω typesetting engine, based on that of TEX, is used to print documents using this font. The characters can be accessed either directly, or through the use of filters, called Ω Typesetting Processes (ΩTPs), which are applied to the input stream.

1 Introduction

Typesetting in different scripts and languages is a problem that is arguably solved today, either by software giants that adapt their products to what they consider to be local typesetting specifications, or by individual users brewing their own limited, but practical, systems for their own needs.

However, typesetting in different scripts and languages—*while still keeping the quality of traditional typography*—and having an open system that can be adapted to any language and script, without loss of power or quality, is still an unachieved goal. The Ω project aims to solve precisely this problem.

The Ω project consists of the design of fonts for different languages and scripts, as well as of an engine that can be used for all possible situations.

As a result, in order to ensure efficiency and openness, one can work at different levels, each one adapted to a specific aspect of multilingual typesetting. These levels correspond to methods used in the Ω system. They will be illustrated through the `omlgc` font, designed for—currently—Latin, Greek, Cyrillic, IPA, Armenian, Georgian and Tifinagh.

This paper was typeset by the Ω engine, using fonts `omlgc` and `omarab`.

2 Ω Methods: an Overview

When developing an Ω typesetting convention for a given language, one can work at the following levels:

1. *The font level*. A font is a container of *glyphs* needed to typeset in a given language. These glyphs may or may not correspond to the graphical units of a script, whether these are called "letters", "ideograms" or otherwise; these glyphs may be parts of graphical units, combinations thereof, or new independent symbols.

The glyphs must be provided to the screen previewer and as well as to the printing engine. Ω itself is not concerned with these glyphs: as with TEX, Ω works only with metrics, not necessarily those for the fonts actually containing the glyphs.

The font level is the lowest development level, in the sense that glyphs are indivisible units that can be used in other higher level structures but cannot be dynamically modified by Ω itself.

2. *The virtual font level*. Once we have the glyphs we need, we combine them to form what is normally considered for a language to be a gramatically correct script entity (a ring accent alone is of no use, but å is part of the grammar for the Swedish language).

A virtual font character is a combination of glyphs taken from several "real" fonts, or of other virtual font characters. In the å example above, the glyphs of the ring and of the letter 'a' can actually be taken from entirely different fonts, in different formats (bitmap, PostScript, TrueType, etc.).

Passing from "real" fonts to virtual fonts is essentially a matter of optimization of storage and memory: taking the seven Greek vowels, the three accents, two spirits, diaeresis, subscript iota and macron/breve diacritics (that makes 16 glyphs) we produce 336 (!) virtual glyphs. Describing a character in a virtual font is hundreds of times smaller than the PostScript code describing a hypothetical similar glyph.

Virtual fonts are directly used by Ω: they can have up to 2^{16} positions and 2^{32} kerning pairs. The files created by Ω can be processed (previewed, printed) by utilities we have adapted; if the user has to use his/her own utilities which are not "16-bit clean", there is a tool to "devirtualize" the files, i.e. replace virtual fonts by the underlying real fonts, and make them as standard as TEX output files.

3. *The ΩTP level*. When Ω reads a document, it first tokenizes it and expands commands and then forwards the data to the typesetting engine. Between these two steps we introduce an arbitrary number of filters, which we call Ω Typesetting Processes (ΩTPs). These are written in a Lex-like syntax, are loaded while reading the document, and can be activated and de-activated dynamically.

A typical example for an ΩTP is contextual analysis of Arabic. Of course, this operation can also been done by macros, say using LATEX commands; but an ΩTP will do it much faster, it will avoid conflicts with other LATEX commands, and it will be much easier to create and configure.

Contextual analysis of Arabic is a typical example of a script property that is low level and that should not use *any* macros or other high-level structures. In the case of the Latin script one would not expect a user to constantly think of 'fi' and 'ff' ligatures and place them manually: it would be very bad strategy to use an "fi ligature command"; in Arabic this property is of the same nature, and hence should remain completely transparent.[1]

But efficiency should not only be limited to speed of typesetting: sometimes it is very important to also optimize the configuration and customization time and effort. A typical example is the management of encodings, whether input or output: by using a universal encoding—which we call it *Unicode++*, since it is a superset of Unicode—as intermediate step for our ΩTPs, every new input encoding requires only a foo → Unicode++ ΩTP and every new font encoding only a Unicode++ → foo one; these are significantly easier to make than if one had to rewrite all processing steps, including contextual analysis and other bells and whistles.

4. *The hyphenation and sorting engines*. Hyphenation and sorting rules are grammatical properties of a language: they have nothing to do with typographical aspects, input methods, font encodings, etc. They only have to be performed on the Unicode++ level, i.e. the most abstract one. Once defined at this level, they can immediately be used with every input and output encoding. Ω hyphenation and sorting engines are still under development: for the time being, Ω does not sort, and hyphenation takes place as in TEX, using the (virtual) font encoding.

1. And undoubtedly this would be the case if the *lingua franca* of computer science was Arabic or Urdu, and not English.

5. *The macro-command level.* LaTeX is a terrific construction, featuring commands at multiple levels: TeX primitives, plain TeX commands, internal LaTeX commands, higher LaTeX commands and environments, and user-defined commands and environments. Our goal is to keep all script-dependent processing independent of the LaTeX macro level; because of this goal, every LaTeX package is usable in any script and language, and both Ω package and LaTeX package developers are able to work independently, while still producing mutually compatible software.

By doing this, we also allow ourselves in the future to use a different typesetting engine than that found in TeX, without having to completely redo our system. The ΩTPs might need modification, but the basic structure would remain the same.

3 Level 1: Designing glyphs

For us, a "glyph" is simply a character from a PostScript, TrueType or Metafont font; so why call it such? Because we want to make the distinction between the "images" and the "combinations of images", the latter of which Ω will use as characters for typesetting.

For our `omlgc` font, the glyphs are produced so that the glyphs for different alphabets look similar. With the exception of Tifinagh, the history of the different alphabets making up this font is strongly interrelated; as such, it has been possible to design a single font with a characteristic feel. For Tifinagh, the script has evolved less than the others, so we are doing creative experimentation—only time will tell if it is successful, i.e. native speakers accept it.

For example, the following two words Ըus Պալասանեաճի are Armenian. Although the traditional Armenian script does not look so rounded, it is considered common practice today to typeset books in this manner, and Armenians are not in the least suprised.

The `omlgc` font is a modern looking font. One question that might arise is what to do with classical versions of these alphabets. For ancient Greek, it is common to use modern typefaces, even though there are typefaces specifically made for ancient Greek, such as Porson or Greek Sans Serif. Can this also be done for Coptic and Slavonic excerpts, without breaking the homogeneity of the surrounding "modern" font? We have tried to do this for Slavonic—including not only letters and accents found in Unicode, but also all the other diacritics and symbols needed for Old Church Slavonic—and our example looks like this: Ѡвѣща̀ е̏й їи̏съ и̂ речѐ е̏й. The result is not particularly exiting to look at, but that is the point: it should fit with the surrounding text, not look "exotic".

Finally, a lot of work has gone into the design of IPA glyphs, which are problematic since many of them are basically font variations (italics, small caps, etc.) of other glyphs, thereby making it a nightmare to design fonts including IPA that have variations.

As we will see in the next section, a big part of the `omlgc` font is obtained by combining a limited number of glyphs. This corresponds to a grammatical reality: it is quite natural to assume that placing diacritics does not affect the shapes of either the base character or the diacritic itself. Often this is true, but there are times when typographical quality requires special shapes.

For example, one can imagine an 'h' with a lower vertical stem, so that the accent of the Esperanto character ĥ is not excessively high. This is really a matter of taste, and we include these kinds of characters as variant forms in our font, and leave the decision whether to use them to the user.

Another similar example comes from the use of diaereses over the Greek vowels iota and upsilon, whether lowercase or uppercase; the distance between the two dots of the diaeresis depends on the letter underneath: compare ϊ, ϋ, Ϊ, Ϋ, while in the Latin script the diaeresis (Umlaut, tréma) always has the same width: ï, ö, ẅ, etc.

4 Level 2: Combining Glyphs into Virtual Fonts

A character of a virtual font can be a combination of characters of other fonts (whether real, in which case we actually have glyphs, or again virtual, in which case we have sub-combinations etc.), of black boxes of arbitrary height and width, and of PostScript code. (Since the latter feature is not implemented in all DVI drivers, one should refrain from using it, at least for the moment.)

We have chosen to work intensely with virtual fonts for two reasons: first because by combining glyphs we can optimize space (and space management is crucial when you deal with 16-bit fonts), and second, to be able to use 16-bit fonts without re-writing all of the world's drivers for TEX's DVI format (the underlying real fonts are 8-bit only, so that a devirtualized 16-bit font becomes 8-bit and can be processed by any decent DVI driver).

Virtual fonts are built using a (portable) Perl script, which reads the configuration file `omlgc.cfg`. This file contains one entry for each character of the virtual font. This line consists of (a) a 4-digit hexadecimal number, designating the character's position in the font, (b) an operator, (c) one or more character names, depending on the operator, (d) a certain number of optional operators and values.

The `N` operator.

The `N` stands for "NAME", and simply means that the string following the operator is the (PostScript) name of one or more characters in some of the fonts loaded. For example,

```
03A5 N Upsilon
```

means that at position 03A5 of the virtual font, we have placed the Greek letter capital Upsilon ϒ. The PostScript names we have used try to be as standard as possible. However, most of the time there is no standard, so we just have to invent names.

The same glyph can be used for several font positions: for example, the Croatian Dze Ð has exactly the same shape as the Icelandic Eth Ð: we use the same glyph, and hence the same PostScript name. Nevertheless we try to optimize the use of glyphs so that one can typeset in one script without necessarily loading the PostScript fonts for other scripts: for example, although the capital Latin 'A' has the same shape as the Greek capital Alpha, we use two different glyphs in two different PostScript fonts, so that when typesetting Greek one can avoid loading the Latin PostScript font. These considerations will be irrelevant when we will be able to use 16-bit monobloc PostScript fonts; for the moment, this is not part of the Adobe Type 1 font specification.

There are several options we can use: `#KRN`, `#KRNLEFT`, `#KRNRIGHT`, `#HOFFSET`, `#VOFFSET`.

The first three concern kerning: we can state that a given character has the same kerning behavior as some other character, which we give by name. For example, ç will be kerned exactly like the letter 'c': everytime there is a kerning pair in the kernings file using 'c' a new kerning pair will be defined, using ç instead of 'c' (and if there is a 'c-c' kerning pair, three new kerning pairs will be defined: 'ç-c', 'c-ç', 'ç-ç'). Sometimes we kern a letter like some given letter on the right and like some other letter on the left: this is typically the case of ligatures: æ will be kerned as 'a' on the left, and as 'e' on the right; in that case, we use the `#KRNLEFT` and `#KRNRIGHT` operators:

```
00C6 N AE #KRNLEFT=A #KRNRIGHT=E
00E6 N ae #KRNLEFT=a #KRNRIGHT=e
0110 N Eth #KRN=D
```

The operators `#HOFFSET` and `#VOFFSET` will offset the glyph without affecting the box of the character.

The `XHAC` and `CHAC` operators.

These operators will place diacritics over letters, which are considered to have the same height as the lowercase letter ‘x’ (x-height) or the one of an LGC uppercase letter (cap-height). The idea is that the height of letters can fluctuate: a round letter, like ‘o’, is slighty higher than a flat letter, like ‘z’, to counterbalance a well-known optical effect. The height of accents over these letters must be the same, even if they aren't exactly of x-height, or cap-height: take for example ó and ú; if we would take the real height of these letters, the accent on the former would be slightly higher than the one on the latter.

How is this accent placed precisely? The Perl utility centers the bounding box of the accent over the bounding box of the letter, with a fixed distance of EPSILON (a global value) between the lower boundary of the accent and either the x-height or cap-height of the font (those two are also global values provided with the utility).

Available options are `#KRN`, `#KRNLEFT`, `#KRNRIGHT`, `#LETTERLIKE`, `#ACCENTLIKE` and `#LETTERREVLIKE`.

The options `#LETTERLIKE` and `#ACCENTLIKE` allow us to use given glyphs, with the metrics of other glyphs. These options are extremely important in certain cases. A typical example is Vietnamese: the letter ‘o with hook’ ơ is significantly wider than the plain ‘o’, nevertheless accents have to be centered on the “o part” of the letter: ớ, ờ, ở, ỡ. This trick allows us to correctly place an accent on the vertical stem of a ‘b’ or an ‘h’: ḃ, ḥ, ḧ; to obtain this result, we simply ask the accent to be placed as if the letter was an ‘l’:

```
0603 CHAC b dot #LETTERLIKE=l
0623 CHAC h dot #LETTERLIKE=l
0627 CHAC h dieresis #LETTERLIKE=l
```

We have the same functionality with accents: using the `#ACCENTLIKE` operator we can place an accent as if it was some other accent. The typical example is again Vietnamese, where there are combined accents ‘circumflex + grave’, ‘circumflex + acute’, which have to be centered with respect to the middle axis of the circumflex (and hence as if there were no acute or grave accent):

```
06D0 CHAC O circumflexacute #KERN=O #ACCENTLIKE=circumflex
06D1 XHAC o circumflexacute #KERN=o #ACCENTLIKE=circumflex
06D2 CHAC O circumflexgrave #KERN=O #ACCENTLIKE=circumflex
06D3 XHAC o circumflexgrave #KERN=o #ACCENTLIKE=circumflex
06D4 CHAC O circumflexhook #KERN=O #ACCENTLIKE=circumflex
06D5 XHAC o circumflexhook #KERN=o #ACCENTLIKE=circumflex
06D6 CHAC O circumflextilde #KERN=O #ACCENTLIKE=circumflex
06D7 XHAC o circumflextilde #KERN=o #ACCENTLIKE=circumflex
```

Another example is Slavonic, with letters such as ꙗ, where the accent has to be placed on the right part of the ligature, as in ꙗ́. This means that we should use the metrics of a given character, justified on the right of our box: this is the rôle of the `#LETTERREVLIKE` operator.

The `ADJ` operator.

This operator allows us to concatenate two characters, using a box with width equal to the sum of the widths of the two boxes, ± the eventual kerning between those characters, and height/depth the maximum height/depth of the two characters. We first wanted to use that operator for the Croatian digraphs ‘Lj’, ‘Nj’, etc. but then decided that the whole idea of having code positions for these digraphs was so silly that we could very well do without. The operator nevertheless proved very useful for cases such as the Greek capital vowels with accent Ά, Έ, Ή, Ί, Ό, Ύ, Ώ.

This operator takes a special option, `#MOVELEFT`, which allows us to overide a kern between two characters and move the second one horizontally, together with its box.

5 Levels 3 and 4: ΩTPs and LaTeX Macros

Once the structure of fonts is well organized, we use ΩTPs for low level script- or language-dependent operations and macros for higher level operations. To give an idea of the power of ΩTPs, consider the following Latin transcription for Berber:

> Tifinagh, d--tira timezwura n .imazighen.
> Llant di tmurt--nnegh dat tira n ta.erabt d--tla.tinit.
> Nnulfant--edd dat .imir n ugellid Masinisen. .Imazighen n
> .imir--en, ttarun--tent ghefi.zra, degg .ifran, ghef .igduren,
> maca tiggti ghef i.zekwan : ttarun fell--asen .isem n umettin,
> d wi--t--ilan, d wayen yexdem di tudert--is akken ur t ttettun
> .ina.tfaren.

This piece of text can be printed using the Latin script, as in:

> Tifinaɣ, d_tira timezwura n imaziɣen. Llant di tmurt_nneɣ dat tira n taɛrabt d_tlaṭinit. Nnulfant_edd dat imir n ugellid Masinisen. Imaziɣen n imir_en, ttarun_tent ɣefiẓra, degg ifran, ɣef igduren, maca tiggti ɣef iẓekwan : ttarun fell_asen isem n umettin, d wi_t_ilan, d wayen yexdem di tudert_is akken ur t ttettun inaṭfaren.

or in the original Tifinagh script, as in:

> ⵜⵉⴼⵉⵏⴰⵖ, ⴷ_ⵜⵉⵔⴰ ⵜⵉⵎⵣⵡⵓⵔⴰ ⵏ ⵉⵎⴰⵣⵉⵖⵢⴻⵏ. ⵍⵍⴰⵏⵜ ⴷⵉ ⵜⵎⵓⵔⵜ_ⵏⵏⴻⵖ ⴷⴰⵜ ⵜⵉⵔⴰ ⵏ ⵜⴰⵄⵔⴰⴱⵜ ⴷ_ⵜⵍⴰⵟⵉⵏⵉⵜ. ⵏⵏⵓⵍⴼⴰⵏⵜ_ⴻⴷⴷ ⴷⴰⵜ ⵉⵎⵉⵔ ⵏ ⵓⴳⴻⵍⵍⵉⴷ ⵎⴰⵙⵉⵏⵉⵙⴻⵏ. ⵉⵎⴰⵣⵉⵖⵢⴻⵏ ⵏ ⵉⵎⵉⵔ_ⴻⵏ, ⵜⵜⴰⵔⵓⵏ_ⵜⴻⵏⵜ ⵖⴻⴼⵉẓⵔⴰ, ⴷⴻⴳⴳ ⵉⴼⵔⴰⵏ, ⵖⴻⴼ ⵉⴳⴷⵓⵔⴻⵏ, ⵎⴰⵛⴰ ⵜⵉⴳⴳⵜⵉ ⵖⴻⴼ ⵉẓⴻⴽⵡⴰⵏ : ⵜⵜⴰⵔⵓⵏ ⴼⴻⵍⵍ_ⴰⵙⴻⵏ ⵉⵙⴻⵎ ⵏ ⵓⵎⴻⵜⵜⵉⵏ, ⴷ ⵡⵉ_ⵜ_ⵉⵍⴰⵏ, ⴷ ⵡⴰⵢⴻⵏ ⵢⴻⵅⴷⴻⵎ ⴷⵉ ⵜⵓⴷⴻⵔⵜ_ⵉⵙ ⴰⴽⴽⴻⵏ ⵓⵔ ⵜ ⵜⵜⴻⵜⵜⵓⵏ ⵉⵏⴰⵟⴼⴰⵔⴻⵏ.

In fact, Berber can also be written right-to-left in the Arabic script (font omarab):

> تيفيناغ، د_تيرا تيمزوّرا ن ئمازيغن. لّانت دي تمورت_نّغ دات تيرا ن تاعرابت د_تلاطينيت. نّولفانت_دّ دات ئمير ن وقْلّيد ماسينيسن. ئمازيغن ن ئميرـن، تّارون_تنت غفيزرا، دڤّ ئفران، غف ئڤدورن، ماشا تيڤّتي غف يزكوان : تّارون فلّ_اسن ئسم ن ومتّين، د ويـت_يلان، د واين يخدم دي تودرت_يس اكّن ور ت تّتّون ئناطفارن.

In the latter case, a font and direction change were necessary. Otherwise, only the output ΩTP needed to be changed for the three cases. Of course, LaTeX macros can be used to encapsulate these changes.

6 Discussion

As this document shows, the omlgc font is now usable for typesetting, using Ω, any language that uses the Latin, Greek, Cyrillic and Tifinagh alphabets, and will soon be ready for the other alphabets in this group.

The omlgc font is not the only multi-script font that has been developed. For example, PLAN 9 [1], Bitstream [2], and Windows [13] all have their own fonts, based on the Unicode encoding.

What does distinguish omlgc from the others is the emphasis that has been placed on typesetting. It is assumed right from the beginning that the set of font positions is much greater than the set of positions in, say, the Unicode definition.

This can best be visualized in Figures 3 and 4, given after the references. For the Greek page, we show the complete set of vowels, with all spirits and breathings, with the macron diacritic, used to denote a long vowel in classical Greek. None of these presentation forms appear in the Unicode specification---hence do not appear in the above fonts---but they are absolutely necessary for typesetting Greek.

Similarly, the Cyrillic page shows all the vowels, with both grave (old style) and acute (new style) accents to place emphasis. Once again, none of these forms are in the Unicode specification, but they are needed for typesetting texts in Russian.

But having these extra forms is of no cost, since they are all virtual characters (hence no extra memory), which can be accessed as needed in a text using the ΩTPs.

We complete the paper with four examples. We begin with three versions (roman, italic, bold) of a Greek text ΠΟΛΛΕΣ ΦΟΡΕΣ ΤΗΝ ΝΎΚΤΑ, by ᾿Ανδρέας ᾿Εμπειρῖκος. We follow with three pages of the `omlgc` font.

References

1. C. A. Bigelow and K. Holmes. The design of a Unicode font. *Electronic Publishing* 6(3):289--305, 1993.
2. Bitstream Inc. Bitstream Cyberbit. `http://www.bitstream.com/cyberbit.htm`.
3. Y. Haralambous and J. Plaice. First applications of Ω: Greek, Arabic, Khmer, Poetica, ISO 10646/Unicode, etc. *TUGboat*, 15(3):344--352, 1994.
4. Y. Haralambous and J. Plaice. Ω, a TEX extension including Unicode and featuring lex-like filtering processes. In Wlodek Bzyl and Tomasz Plata-Przechlewski, editors, *Proceedings of the European TEX Conference*, pages 153--166, Gdansk, Poland, 1994. GUST.
5. Y. Haralambous and J. Plaice. ΩTimes and ΩHelvetica Fonts under Development: Step One. *TUGboat*, 17(3):126--146, 1996.
6. Y. Haralambous and J. Plaice. A font for typesetting large mathematical symbols in Ω. *EuroTEX'98*, Saint-Malo, France, 29--31 March 1998.
7. J. Plaice and Y. Haralambous. Typesetting French, German and English in Ω. *EuroTEX'98*, Saint-Malo, France, 29--31 March 1998.
8. Y. Haralambous, J. Plaice, and J. Braams. Never again active characters! Ω-Babel. *TUGboat*, 16(4):418--427, 1995.
9. Y. Haralambous, J. Plaice, and A. Picheral (trans.). Ω, une extension de TEX incluant UNICODE et des filtres de type Lex. *Cahiers GUTenberg*, 20:55--80, 1995. Translation of refereed conference publication [4].
10. J. Plaice. Language-dependent ligatures. *TUGboat*, 14(3):271--274, 1993.
11. J. Plaice. Progress in the Omega project. *TUGboat*, 15(3):320--324, 1994.
12. J. Plaice and Y. Haralambous. The latest developments in Ω. *TUGboat*, 17(3):181--183, 1996.
13. Microsoft Inc. True Type in Windows 95. `http://www.microsoft.com/truetype/tt/win95tt.htm`.

ΠΟΛΛΕΣ ΦΟΡΕΣ ΤΗΝ ΝΥΚΤΑ
᾽Ανδρέας ᾽Εμπειρῖκος

῞Οσοι ἀπὸ σᾶς γυρίζετε τὴν νύκτα μέσ᾽ στοὺς δρόμους, ἀμέριμνοι ἢ σκεπτικοί, τὴν ἄνοιξι, κατὰ τὴν ἐποχὴ τοῦ ᾽Επιταφίου Θρήνου, ἢ ἐκεῖ κοντὰ στὶς ὧρες τὶς χαρούμενες ποὺ ὁδηγοῦν στὴν θριαμβευτικὴν τὴν ἄνωσιν ποὺ πάει νὰ γίνῃ Πάσχα, πρὶν ἀκουσθοῦν οἱ ἀναστάσιμες καμπάνες, καί, ἀκόμη περισσότερο, τὶς νύκτες τοῦ καλοκαιριοῦ στοὺς δρόμους τοὺς ὀνειρικοὺς τοῦ σκοτεινοῦ Λονδίνου, στοὺς ἄλλους τοὺς πλατεῖς ἢ τοὺς στενούς, ποὺ ἐκτείνονται γύρω ἀπὸ τὸν Μόσχοβα στὴ Μόσχα, ἢ στὰς ὁδοὺς τῆς κάτασπρης ᾽Αθήνας, σέ δορυάλωτες στιγμὲς τῆς θλίψεως, ἢ σὲ ἀφρόεσσες στιγμὲς εὐδαιμονίας, ὅταν παράθυρα καὶ ἐξώφυλλα χαίνουν διάπλατα ἀνοικτὰ γιὰ νὰ δεχθοῦν δροσιὰ καὶ μῦρα, ὅσοι ἀπὸ σᾶς νύκτωρ γυρίζετε στοὺς δρόμους πανευτυχεῖς ποὺ ἐκσπερματίσατε, ἢ δυστυχεῖς ποὺ κάποια γυναῖκα δὲν ἔστερξε νὰ σᾶς δεχθῇ καὶ δὲν ἐστάθη, λίγο ἂν προσέξετε, θὰ ἀκούσετε πολλά, ὅσα στήν τύρβη τῆς ἡμέρας δύσκολον εἶναι νὰ ἀκουσθοῦν.

῞Οσοι ἀπὸ σᾶς γυρίζετε τὴν νύκτα μέσ᾽ στοὺς δρόμους, ἀμέριμνοι ἢ σκεπτικοί, τὴν ἄνοιξι, κατὰ τὴν ἐποχὴ τοῦ ᾽Επιταφίου Θρήνου, ἢ ἐκεῖ κοντὰ στὶς ὧρες τὶς χαρούμενες ποὺ ὁδηγοῦν στὴν θριαμβευτικὴν τὴν ἄνωσιν ποὺ πάει νὰ γίνῃ Πάσχα, πρὶν ἀκουσθοῦν οἱ ἀναστάσιμες καμπάνες, καί, ἀκόμη περισσότερο, τὶς νύκτες τοῦ καλοκαιριοῦ στοὺς δρόμους τοὺς ὀνειρικοὺς τοῦ σκοτεινοῦ Λονδίνου, στοὺς ἄλλους τοὺς πλατεῖς ἢ τοὺς στενούς, ποὺ ἐκτείνονται γύρω ἀπὸ τὸν Μόσχοβα στὴ Μόσχα, ἢ στὰς ὁδοὺς τῆς κάτασπρης ᾽Αθήνας, σέ δορυάλωτες στιγμὲς τῆς θλίψεως, ἢ σὲ ἀφρόεσσες στιγμὲς εὐδαιμονίας, ὅταν παράθυρα καὶ ἐξώφυλλα χαίνουν διάπλατα ἀνοικτὰ γιὰ νὰ δεχθοῦν δροσιὰ καὶ μῦρα, ὅσοι ἀπὸ σᾶς νύκτωρ γυρίζετε στοὺς δρόμους πανευτυχεῖς ποὺ ἐκσπερματίσατε, ἢ δυστυχεῖς ποὺ κάποια γυναῖκα δὲν ἔστερξε νὰ σᾶς δεχθῇ καὶ δὲν ἐστάθη, λίγο ἂν προσέξετε, θὰ ἀκούσετε πολλά, ὅσα στήν τύρβη τῆς ἡμέρας δύσκολον εἶναι νὰ ἀκουσθοῦν.

῞Οσοι ἀπὸ σᾶς γυρίζετε τὴν νύκτα μέσ᾽ στοὺς δρόμους, ἀμέριμνοι ἢ σκεπτικοί, τὴν ἄνοιξι, κατὰ τὴν ἐποχὴ τοῦ ᾽Επιταφίου Θρήνου, ἢ ἐκεῖ κοντὰ στὶς ὧρες τὶς χαρούμενες ποὺ ὁδηγοῦν στὴν θριαμβευτικὴν τὴν ἄνωσιν ποὺ πάει νὰ γίνῃ Πάσχα, πρὶν ἀκουσθοῦν οἱ ἀναστάσιμες καμπάνες, καί, ἀκόμη περισσότερο, τὶς νύκτες τοῦ καλοκαιριοῦ στοὺς δρόμους τοὺς ὀνειρικοὺς τοῦ σκοτεινοῦ Λονδίνου, στοὺς ἄλλους τοὺς πλατεῖς ἢ τοὺς στενούς, ποὺ ἐκτείνονται γύρω ἀπὸ τὸν Μόσχοβα στὴ Μόσχα, ἢ στὰς ὁδοὺς τῆς κάτασπρης ᾽Αθήνας, σέ δορυάλωτες στιγμὲς τῆς θλίψεως, ἢ σὲ ἀφρόεσσες στιγμὲς εὐδαιμονίας, ὅταν παράθυρα καὶ ἐξώφυλλα χαίνουν διάπλατα ἀνοικτὰ γιὰ νὰ δεχθοῦν δροσιὰ καὶ μῦρα, ὅσοι ἀπὸ σᾶς νύκτωρ γυρίζετε στοὺς δρόμους πανευτυχεῖς ποὺ ἐκσπερματίσατε, ἢ δυστυχεῖς ποὺ κάποια γυναῖκα δὲν ἔστερξε νὰ σᾶς δεχθῇ καὶ δὲν ἐστάθη, λίγο ἂν προσέξετε, θὰ ἀκούσετε πολλά, ὅσα στήν τύρβη τῆς ἡμέρας δύσκολον εἶναι νὰ ἀκουσθοῦν.

Figure 1. Roman, italic and bold versions of a Greek text

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0600	Ạ	ạ	Ḃ	ḃ	Ḅ	ḅ	Ḇ	ḇ	Ḉ	ḉ	Ḋ	ḋ	Ḍ	ḍ	Ḏ	ḏ
0610	Ḑ	ḑ	Ḓ	ḓ	Ḕ	ḕ	Ḗ	ḗ	Ḙ	ḙ	Ḛ	ḛ	Ḝ	ḝ	Ḟ	ḟ
0620	Ḡ	ḡ	Ḣ	ḣ	Ḥ	ḥ	Ḧ	ḧ	Ḩ	ḩ	Ḫ	ḫ	Ḭ	ḭ	Ḯ	ḯ
0630	Ḱ	ḱ	Ḳ	ḳ	Ḵ	ḵ	Ḷ	ḷ	Ḹ	Ḻ	Ḽ	ḽ	Ḿ	ḿ		
0640	Ṁ	ṁ	Ṃ	ṃ	Ṅ	ṅ	Ṇ	ṇ	Ṉ	ṉ	Ṋ	ṋ	Ṍ	ṍ	Ṏ	ṏ
0650	Ṑ	ṑ	Ṓ	ṓ	Ṕ	ṕ	Ṗ	ṗ	Ṙ	ṙ	Ṛ	ṛ	Ṝ	ṝ	Ṟ	ṟ
0660	Ṡ	ṡ	Ṣ	ṣ	Ṥ	ṥ	Ṧ	ṧ	Ṩ	ṩ	Ṫ	ṫ	Ṭ	ṭ	Ṯ	ṯ
0670	Ṱ	ṱ	Ṳ	ṳ	Ṵ	ṵ	Ṷ	ṷ	Ṹ	ṹ	Ṻ	ṻ	Ṽ	ṽ	Ṿ	ṿ
0680	Ẁ	ẁ	Ẃ	ẃ	Ẅ	ẅ	Ẇ	ẇ	Ẉ	ẉ	Ẋ	ẋ	Ẍ	ẍ	Ẏ	ẏ
0690	Ẑ	ẑ	Ẓ	ẓ	Ẕ	ẕ	ẖ	ẗ	ẘ	ẙ	aʾ	ẛ				
06A0	Ạ	ạ	Ả	ả	Ấ	ấ	Ầ	ầ	Ẩ	ẩ	Ẫ	ẫ	Ậ	ậ	Ắ	ắ
06B0	Ằ	ằ	Ẳ	ẳ	Ẵ	ẵ	Ặ	ặ	Ẹ	ẹ	Ẻ	ẻ	Ẽ	ẽ	Ế	ế
06C0	Ề	ề	Ể	ể	Ễ	ễ	Ệ	ệ	Ỉ	ỉ	Ị	ị	Ọ	ọ	Ỏ	ỏ
06D0	Ố	ố	Ồ	ồ	Ổ	ổ	Ỗ	ỗ	Ộ	ộ	Ớ	ớ	Ờ	ờ	Ở	ở
06E0	Ỡ	ỡ	Ợ	ợ	Ụ	ụ	Ủ	ủ	Ứ	ứ	Ừ	ừ	Ử	ử	Ữ	ữ
06F0	Ự	ự	Ỳ	ỳ	Ỵ	ỵ	Ỷ	ỷ	Ỹ	ỹ						

Figure 2. Unicode Row 1E: Latin Extended Additional

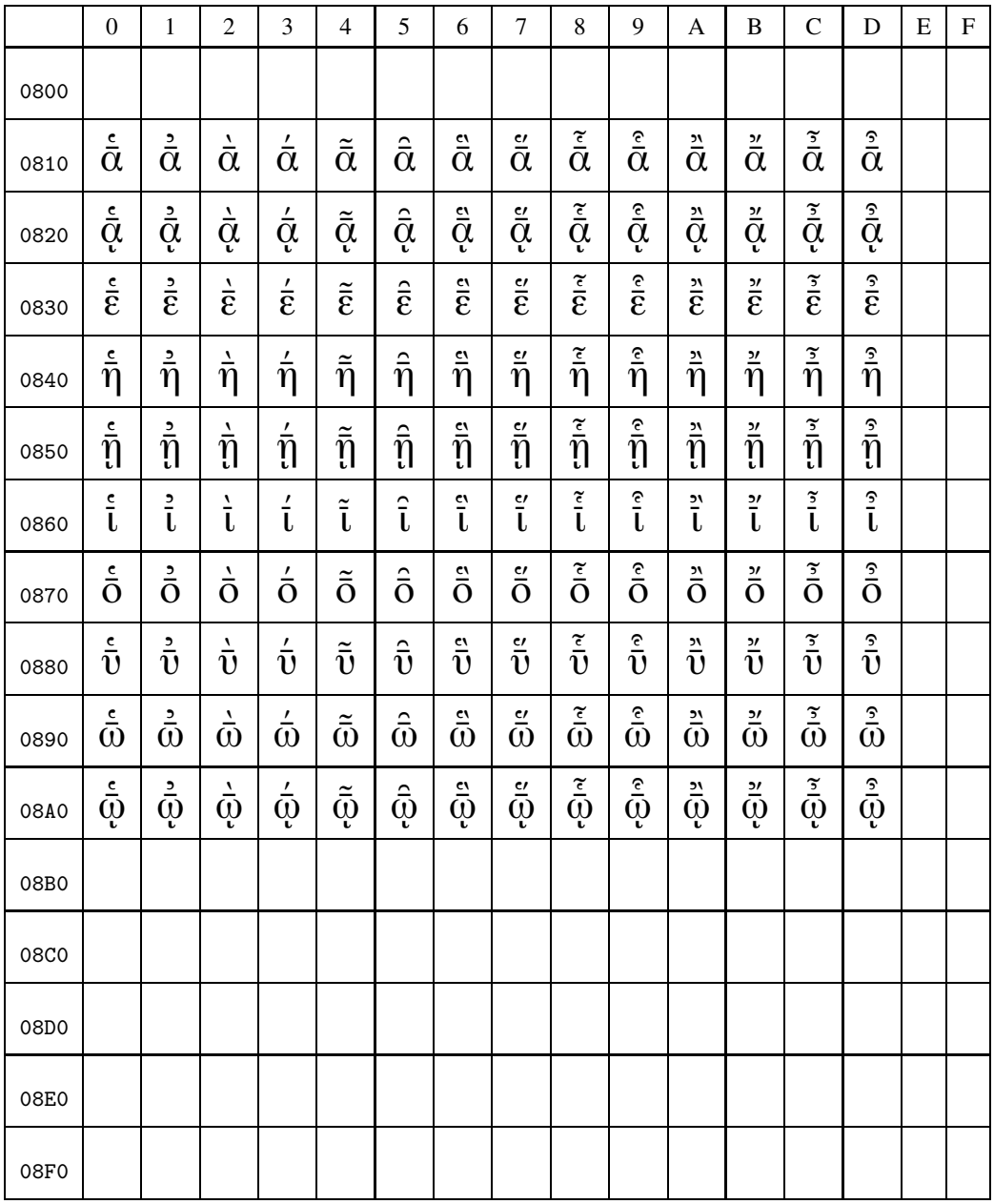


Figure 3. Greek vowels with macron, for long syllables (not in Unicode)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0A00	Г	г	Ӂ	ӂ	Ӡ	ӡ	Қ	қ	Ң	ң	Ҫ	ҫ	Ҳ	ҳ	Ҭ	ҭ
0A10	Ӆ	ӆ	Ӆ	ӆ	Ӊ	ӊ	Ҭ	ҭ	Ӌ	ӌ	Ā	ā	Ō	ō	Ọ	ọ
0A20	Ў̓	ў̓	Ѝ	ѝ	Ӯ	ӯ	Ӯ	ӯ	Э̄	э̄	Э̇	э̇	Ю̄	ю̄	Я̄	я̄
0A30	Ӏ	ӏ	І	і	Ө̄	ө̄	Ӯ	ӯ	Ѓ	ѓ	Ѐ	ѐ	Ё̋	ё̋	Ѓ	ѓ
0A40	Ӥ	ӥ	Ї	ї	Ї̋	ї̋	Ӧ	ӧ	Ӱ	ӱ	Ы	ы	Ъ́	ъ́	Э́	э́
0A50	Ю́	ю́	Я́	я́	Ъ́	ъ́	Ѵ	ѵ	А̀	а̀	Ӈ́	ӈ́	Ӂ́	ӂ́	Ӈ́	ӈ́
0A60	Ӄ	ӄ	Ӱ	ӱ	Ѹ	ѹ	Ӈ́	ӈ́	Ѡ	ѡ	Ѻ	ѻ	Ӳ	ӳ	Ӕ	ӕ
0A70	Э́	э́	Ө́	ө́	Ӳ	ӳ	Ъ́	ъ́								
0A80	А̀	Ӈ̀	А̀	Ѐ	Ӄ	Ӥ	Ѝ	Ѻ̀	Ѡ̀	Ю̀	Ӱ̀	Ѵ	Ъ̀			
0A90	а̀	ӈ̀	а̀	ѐ	ӄ	ӥ	ѝ	ѻ̀	ѡ̀	ю̀	ѹ̀	ѵ	ъ̀			
0AA0	А̂	Ӈ̂	А̂	Ѐ̂	Ӄ̂	Ӥ̂	Ѝ̂	Ѻ̂	Ѡ̂	Ю̂	Ӱ̂	Ѵ̂	Ъ̂			
0AB0	а̂	ӈ̂	а̂	ѐ̂	ӄ̂	ӥ̂	ѝ̂	ѻ̂	ѡ̂	ю̂	ӱ̂	ѵ̂	ъ̂			
0AC0	А̋	Ӈ̋	А̋	Ѐ̋	Ӄ̋	Ӥ̋	Ї̋	Ѻ̋	Ѡ̋	Ю̋	Ӱ̋	Ѵ̋	Ъ̋			
0AD0	а̋	ӈ̋	а̋	ѐ̋	ӄ̋	ӥ̋	ї̋	ѻ̋	ѡ̋	ю̋	ӱ̋	ѵ̋	ъ̋			
0AE0	А̏	Ӈ̏	А̏	Ѐ̏	Ӄ̏	Ӥ̏	Ї̏	Ѻ̏	Ѡ̏	Ю̏	Ӱ̏	Ѷ	Ъ̏			
0AF0	а̏	ӈ̏	а̏	ѐ̏	ӄ̏	ӥ̏	ї̏	ѻ̏	ѡ̏	ю̏	ӱ̏	ѷ	ъ̏			

Figure 4. Accented Cyrillic characters, for emphasis (not in Unicode)

language and fonts

TEX in Polish

Erik Frambach
Faculty of Economics
University of Groningen
E.H.M.Frambach@eco.rug.nl

abstract

Writing in Polish with TEX requires a few tricks. In Polish you need several accents that are often not available in 'standard' fonts. Some TEX macros can solve this problem more or less. We will show the pros and cons.
Another 'problem' is input encoding. One can use 8-bit input in combination with the corresponding codepage definition, or a 7-bit encoding with a few extras to make typing easier. Both methods will be discussed.
This article reflects the content of a lecture held at the NTG meeting on 11 November 1999.

keywords

Polish, ogonek, input encoding

Dzień dobry!

Any idea what the following text means?

> Serdecznie dziękuje nie ma za co, proszę. Przepraszam, do widzenia do zobaczenia na razie nie rozumiem. Jak to się, mówi po polsku? Jak masz na imię jak się pani nazywa miło mi cię. Pana panią poznać jak się masz jak się pan ma dobrze. Źle tak sobie jako tako gdzie jest toaleta?

Actually, it's rubbish. I just combined some standard phrases into something that *looks* like a plain paragraph. But, anyway, the listing below will give you some idea of the meaning of some phrases I abused.

- Dzień dobry = Good day
- Serdecznie dziękuje = Thank you
- Nie ma za co, Proszę = You're welcome
- Przepraszam = Excuse me
- Do widzenia; do zobaczenia = See you later, goodbye
- Na razie = Farewell, bye
- Nie rozumiem = I don't understand
- Jak to się, mówi po polsku? = How do you say this in Polish?
- Jak masz na imię? Jak sie, Pani nazywa? = What's your name?
- Miło mi cię (pana, panią) poznać = Nice meeting you
- Jak się, masz? Jak się pan ma? = How are you?
- Dobrze = Good
- Źle = Bad
- Tak sobie; Jako tako = Alright
- Gdzie jest toaleta? = Where is the toilet?

Polish problems

When writing Polish text in TEX we have to deal with the following problems:

- TEX accents:

The *acute* accent is used a lot: e.g. ń, ś, ć, ó. The *dot* accent is also used in Polish: ż. Both are readily available in all fonts and can be accessed using the standard TEX commands `\'` and `\.` respectively. These accents are acceptable but could be improved. E.g., the dot accent may seem too thick and the `\'` accent should be flattened slightly when applied to capitals. There is a Polish version of the Computer Modern fonts that implements these changes. Instead of `cm*.*` they are called `pl*.*` and they are fully compatible with the Computer Modern fonts. Both PL fonts and EC fonts use 'Cork encoding' so they are compatible with respect to Polish diacritics.

- Non-TEX accents:

The *ogonek* accent is used on several characters. It's the small 'tail' in e.g. ę and should not be confused with the *cedilla* that looks like this: ę. The ogonek is more difficult to typeset because many fonts do not support it. So we have to either make a kludge or extend the fonts we use with extra characters. LATEX users can use the `ogonek` package. It defines a macro `\k` that will emulate an ogonek on its parameter. The results, however, are not optimal, to say the least. A better solution is to use 'Polished' fonts. The EC fonts also contain true ogonek characters. Below is a comparison of Computer Modern using the LATEX `ogonek` package, the Polish version of the Computer Modern fonts, and Times Roman using the LATEX `ogonek` package and the real ogonek accent provided in the font:

Computer Modern (ogonek): ęĄą

Computer Modern (Polish): ęĄą

Times Roman (ogonek): ęĄą

Times Roman (real): ęĄą

□ Special characters:
In Polish you need the 'l-slash' and the 'L-slash'. Most fonts have these characters, and they can be accessed by the standard TEX commands `\l` and `\L`. However, the CM versions of these character are hardly acceptable in Polish. In the Polish versions the angle and the length of the bar are different:

CM Roman: ł Ł

PL roman: ł Ł

□ Hyphenation:
Because of the many accents in Polish texts hyphenation can be a problem. If you simply use standard TEX commands for accents hyphenation will not work properly. That is, hyphenation will not be incorrect but many valid hyphenation points will not be considered. 8-bit encoded hyphenation patterns in combination with 8-bit input (or emulated 8-bit input) can solve this problem.

Polish notation

The Polish are already famous for their 'reverse Polish notation' so why not build another one. Instead of using backslashes it's more convenient to use forward slashes for writing Polish text (you could say the backslash is 'reversed'!).

In Polish formats the slash is actually a character that can be 'activated' and 'deactivated' using `\prefixing` and `\nonprefixing`. When active the slash will take care of applying the correct accent to the following character. In Polish it's always clear which one it should be, unlike in e.g. French or Dutch. So, `/s` will produce ś, `/z` will produce ż and `/e` will produce ę. This method works for Polish (and not for e.g. French) because in Polish there can be no confusion over which accent should be used. An 'e' can only become ę, not e.g. ë, etc. There is just one exception to the rule: ź. This one can be coded as `/x`. Uppercase variants also exist. Naturally the slash macro will also make sure that hyphenation works properly. Very neat.

Since the slash notation is so powerful, why not use it in hyphenation patters as well? Here is a snippet of the Polish hyphenation patterns that shows how the slash is made active and defined as a macro that outputs an accented character based on its parameter.

```
\catcode`\/=13
\def/#1{%
  \ifx#1a^^a1\else
  \ifx#1c^^a2\else
  \ifx#1e^^a6\else
  \ifx#1l^^aa\else
  \ifx#1n^^ab\else
  \ifx#1o^^f3\else
  \ifx#1s^^b1\else
  \ifx#1x^^b9\else
  \ifx#1z^^bb%
  \fi\fi\fi\fi\fi\fi\fi\fi\fi}%
```

Lowercase codes are provided in order to make the hyphenation patters work for these accented characters:

```
\lccode "A1 = "A1 % /a (161)
\lccode "A2 = "A2 % /c (162)
\lccode "A6 = "A6 % /e (166)
\lccode "AA = "AA % /l (170)
\lccode "AB = "AB % /n (171)
\lccode "F3 = "F3 % /o (243)
\lccode "B1 = "B1 % /s (177)
\lccode "B9 = "B9 % /x (185)
\lccode "BB = "BB % /z (187)
```

Now the slash notation can be used in hyphenation patterns:

```
\patterns{
./c/c8
./c/l8
./c/n8
./c/s8
./c/z8
./c8
./cb8
.b/c8
.b/l8
.b/n8
.b/s8
.bc8
.bd8
.be2z3
.be3z4an
.ca/lo3/s2
.ca/lo3k2
.nad/srod5ziem
.zado/s/cu4
.zado2/s/c3
po/lu3d2ni
```

```
sze4/s/c
...
```

Input encoding

For Polish people using MS-Windows it's natural to use codepage 1250 (East European) to input text because it supports all the characters they need. On Unix ISO8859-2 is used, and on MS-DOS and Atari it's usually codepage 852. On Amiga and Macintosh machines you may find still other encodings! However, the LaTeX `inputenc` package can be used to process such text without change. You only need to include the following statement in the preamble of the LaTeX document:

```
\usepackage[cp1250]{inputenc}
```

The `inputenc` package will map special characters to the appropriate macros. Below is a small part of `cp1250.def` that shows how this is done. Note how the `\k` macro is used for ogonek.

```
\DeclareInputText{156}{\@tabacckludge's}
\DeclareInputText{163}{\L}
\DeclareInputText{165}{\k A}
\DeclareInputText{175}{\.Z}
\DeclareInputText{179}{\l}
\DeclareInputText{185}{\k a}
\DeclareInputText{186}{\c s}
```

Encoding translation tables

Another way of dealing with the East European codepage is to use an *encoding translation table*, a feature supported by some TeX implementations such as emTeX and Web2c (teTeX, fpTeX) and MikTeX. In that case the input is translated from one encoding into another one *before* TeX reads it. This can be very convenient but one must be aware that files that depend on this feature may not be processed correctly on other TeX systems. Below is a snippet of such a translation file:

```
%% cp1250pl.tcx:
%%   encoding translation table for TeX
%% source (TeX input):
%%   cp1250 (Windows East European)
%% target (TeX intestines):
%%   PL and QX encoding (Polish PL and QX fonts)
%
%% MAIN ENCODING TABLE:
...
0x9c 0xb1 % 156 177 sacute
0x9e 0xba % 158 186 zcaron
0x9f 0xb9 % 159 185 zacute
0xa3 0x8a % 163 138 Lslash
0xa5 0x81 % 165 129 Aogonek
...
```

An encoding translation table can used in two ways. The TeX compiler can be invoked with a parameter like this:

```
tex -translate-file=cp1250pl myfile
```

You can also add this parameter to the file itself. It must be a *comment* on the first line like this:

```
%& -translate-file=cp1250pl
```

Integration

Now that we have solutions to the basic problems for Polish TeX we can integrate them to make (TeX) life easier.

- The Polish have defined their own variant of plainTeX called MeX. It does everything that plain TeX does but also supports the slash notation and uses PL fonts. It also gives access to French style guillemots used in nested citations, and it supports repetition of the hyphen character on the beginning of the next line when breaking a paragraph into lines.
- A restricted LaTeX format was made in which only US language and Polish language is supported, without using Babel: it uses `polski.sty`. Naturally the PL fonts are used by default and EC fonts are supported using `\usepackage[T1]{fontenc}`. The `polski` package can also used with standard LaTeX if you want to use the Polish hyphenation patterns.
- A Polish version of the MakeIndex program was produced because the accented characters require a more complex sorting algorithm. This version can be adopted to other 8-bit encodings as well.
- An 8-bit version of the BibTeX program was produced. It knows how to deal with the Polish accented characters.

References

A detailed report called 'Polishing TeX: from ready to use to handy in use' by Bogusław Jackowski and Marek Ryćko was presented at the EuroTeX'92 conference in Prague. Staszek Wawrykiewicz made various suggestions to improve this article.

Do widzenia!

Goodbye!

TEXniques
Toolbox

Maarten Gelderman

abstract

This toolbox contains some varia. First I discuss some reactions on remarks I made in an earlier toolbox. Next, Hans Hagens texexec is used in the following section to create eps and pdf files from metapost source. Files created this way are often more usefull than the eps files metapost itself creates. How to prepare a single source file for usage with both traditional TEX and pdfTEX is discussed next. I also show how easy it is to set up a font different from Computer Modern for typesetting simple mathematics, pay some attention to a failed attempt to install a TrueType font and present a small PostScript header file that can be used to produce watermarks.

keywords

make, texexec, pdf, eps, math fonts

The previous toolbox

Two persons gave comments and corrections related to the previous toolbox. Jules van Weerden informed me that my description of make was flawed. In the previous toolbox I indicated that if multiple makefiles exist, make will first try to use the one capitalized as Makefile. Of course, it is never prudent to depend on capilization for such decisions, however, in this case it may even turn out to be lethal: the behaviour described by me is not standard make behavior, so any one using another make than that of the GNU variety may get unexpected results.

Wybo Dekker gave an improvement of my unreadable (emacs) regular expression. I presented the next regexp: `\\\(['\|\^\|'\|"]\)\([^\{]\)` → `\\\1{\2}`, Wybo mailed me the shorter—but fortunately equally unreadable—alternative: `\(\\['^'"]\)\([^\{]\)` → `\1{\2}`. But of course, real men, like Wybo, do not use emacs, but Perl. For those interested in carrying out the same substitution using this program, the regexp to use is: `s/(\\['^'"])([^{])/$1\{$2}/g`.

Making makefiles superfluous

However, since Hans Hagens texexec is available makefiles are superfluous to the average TEX-user. This Perl-script, which I will not discuss here, just read Hans' own description of it, takes care of all stages of compilation. It can even be used for easy previewing of metapost-files. By issuing the following command `texexec -output=pdftex -figures=c -tex=cont-en -result=plaatjes plaatjes.[0-9]*`, a PDF-file will be created which contains the pictures generated from the metapost-file plaatjes.mp.[1] The ability to generate pdf-files from metapost generated pictures is not only handy for previewing. It also is a reasonable foolproof way to solve font conflicts. Metapost does not include the fonts it uses in the eps-files it generates. It depends on a postprocessor (most often dvips or pdfTEX) for the final inclusion of these fonts. Although this approach may have some advantage in terms of efficiency, conflicts involving fontnames all too often occur. The pdf-file generated in the way described above is really self-contained and consequently solves such ambiguities. If desired, you can use GhostScript (pdf2ps) to convert the pdf file to (encapsulated) postscript.

Of course, I find typing the whole texexec command too much work, so I made yet another Makefile, to simplify life:

```
plaatjes.1: plaatjes.mp
        export TEX=latex;mpost factor
        texexec --output=pdftex --figures=c\
        --tex=cont-en --result=plaatjes    \
        plaatjes.[0-9]*
```

Using the same document with ordinary TEX and pdfTEX

Most readers of this journal will know it by now. TEX can be used to make (nearly) perfect PDF-documents. When you use the pdfTEX executable instead of the ordinary TEX, PDF-files instead of DVI-files are easily generated by executing the command `\pdfoutput=1` somewhere at the start of your document. LATEX users can obtain more advanced results by using packages like hyperref (the `\pdfoutput=1` is superfluous when this package is used). If you for instance include `\usepackage [pdftex, bookmarks=true, bookmarksopen=true] {hyperref}`

1. The `[0-9]`-convention used in the command requires you to use a reasonably modern shell like bash. Windows-users will probably have to enter the names of the metapost-generated files by hand.

in your preamble, bookmarks will be generated automatically and will be visible when the document is opened in Acrobat Reader.

However, this approach introduces one problem: every now and then you will want to use your ordinary TEX-executable to process this document. If your document includes the commands mentioned above, TEX will issue an error-message: either these commands themselves are undefined, or they make use of undefined commands. The solution to this problem capitalizes on this same feature: if \pdfoutput is defined, we know we are using pdfTEX and simply assume we want to generate a PDF-file. If it is undefined, apparently a traditional TEX is being used and we do not want to issue these PDF related commands. The implementation is rather simple. First we define a new boolean variable:

```
\newif\ifpdf
```

Next we check whether \pdfoutput is defined. If this is not the case we do not set \pdfoutput to 1 and we assign a false value to our boolean. In other cases we assign a value to \pdfoutput and make our boolean true.[2]

```
\ifx\pdfoutput\undefined
  \pdffalse
\else
  \pdfoutput=1
  \pdftrue
\fi

\ifpdf
  \usepackage[pdftex]{graphicx}
  \graphicspath{{pdf/}{png/}}
\else
  \usepackage[dvips]{graphicx}
  \graphicspath{{eps/}}
\fi
```

In the code presented above the \ifpdf boolean is also used to solve another problem: in ordinary TEX I traditionally use .eps-files. PdfTEX is only able to process .eps-files generated by metapost. Fortunately pdfTEX *is* able to deal with two other file formats: PNG (portable network graphics, for bitmap files) and PDF. Assuming that each picture used in our document is present in .eps-format in the eps-directories and in either .png or .pdf format in the png and pdf directory respectively, the above code assures that depending on the executable used, the right graphics-file will be selected.

Poor-man's-math

By now, most TEX users are aware of the fact that we do not need to stick to Computer Modern when preparing our documents. As long as one does not want to typeset mathematics, Type I fonts are easily integrated. If one wants to do advanced mathematical typesetting, using Computern Modern remains the only option for most users. The reason traditionally given is simple: a whole bestiary of mathematical symbols is available in Computer Modern. This same bestiary—and more often only a subset of it—can only be found in a number of extensions to commercial fonts. Such extensions are available for Times, Helvetica, Courier and Lucida. The user who does not want to buy these fonts, is forced to stick to Computer Modern, or to combine his/her Type I font with Computer Modern or Euler (a free mathematics only font).[3]

So far for theory, now to practice. I do not know how your documents look, but the math in a lot of my documents is hardly more complex than $y = a + b_1x_1 + b_2x_2$. Why would I need complex math symbols for such documents? Provided that I can live with a limited set of symbols and would be willing to sacrifice some of the niceties of math spacing that TEX normally tries to provide, shouldn't it be possible to find some way to set such simple math out of a arbitrary Type-I font? After some experiments with commands discussed in *The LaTeX companion* I learned that, although TEX complains bitterly about such indecent treatment, the next set of commands produces reasonably acceptable results.

```
\documentclass{article}
\renewcommand{\rmdefault}{padj}
\DeclareMathVersion{normal}
\DeclareMathVersion{bold}
\DeclareSymbolFont{operators}{OT1}{pad}{m}{n}
\SetSymbolFont{operators}{bold}{OT1}{pad}{b}{n}
\DeclareSymbolFont{letters}{OT1}{padj}{m}{it}
\SetSymbolFont{letters}{bold}{OT1}{padj}{b}{it}

\begin{document}\thispagestyle{empty}
$$\sum_{a=1}^{\infty}\frac{a_1+b^2}{3+c^{i_3}}$$
\end{document}
```

I do not claim to know exactly what I am doing, so I can only give a small elaboration on the meaning of these commands. The main thing to notice is the padj and pad stuff, which stand for Adobe Garamond with and without oldstyle numerals respectively. If you replace both pad and padj by e.g. phv your formulas will be set in Helvetiva, if you replace both by pcr, Courier will be used instead etc.

2. Some LaTeX-classes/packages define \pdfoutput, consequently it is essential to set the boolean to false or true before any classes/packages are loaded.

3. For Times, an alternative is available: the package mathptm does its uttermost best to imitate a Times with additional math symbols using a combination of Times, the PostScript Symbol font and Computer Modern.

$$\sum_{a=1}^{\infty} \frac{a_1 + b^2}{3 + c^{i_3}} \qquad \sum_{a=1}^{\infty} \frac{a_1 + b^2}{3 + c^{i_3}} \qquad \sum_{a=1}^{\infty} \frac{a_1 + b^2}{3 + c^{i_3}}$$

Figure 1. Examples of poor-man's-math as described in this article, using Adobe Garamond, Helvetica and Courier respectively.

A disillusion

Being able to use Type I fonts of course is nice. But of some fonts I happen to own just a TrueType version. Conceptually using these fonts does not seem to be too difficult. A first possible solution is conversion from TrueType to Type 1. However, this will result in lower quality font. A second approach seems more worthwhile: more recent PostScript interpreters can deal with so-called Type 42 fonts. Type 42 fonts can be used to 'embed' TrueType fonts in a PostScript font. This did seem to be the way to go. However, although I got a lot of help from the people on TEX-NL, I was not able to do the conversion. The AFM-files generated by the conversion program do not contain information on the x-height and the cap-height of the font and TEX can't do without. I hope I can report some progress in a next edition of MAPS.

Watermarks

Some people apparently are not able to live without them anymore: watermarks. On every sheet of paper that leaves their hands they print their name or some other superfluous remark in the background. I hate those watermarks, but nevertheless one sometimes does need them. Recently I had to present the results of a research project to the scientific council of a research institute. I was warned beforehand that it would be prudent to take some measures that would ensure that my instrument could not be copied without my name on it. 'You never know where those copies end up, and people might just not remember how they obtained them.' One does not ignore such advise. Fortunately I did remember that Siep Kroonenberg posted some information on this topic on TEX-NL, the information she provided, combined with the *PostScript Language Tutorial and Cookbook* sufficed to make the following code:

```
%!
/bop-hook {gsave
  /Helvetica-Bold findfont 35 scalefont setfont
  .7 setgray
  144 72 moveto 90 rotate
  gsave
  (If you want to use this instrument) show
  grestore
  0 -40 rmoveto
  (please contact Maarten Gelderman) show
  grestore} def
```

This little piece of PostScript code (which I saved in a file called `watermark.pro`) defines the begin-of-page (bop) hook. The graphic state of the PostScript interpreter is saved at the beginning, and restored at the end. The first thing the header file does is looking for a bold Helvetica, scaling it to 35 points and selecting it as the current font (PostScript uses Reverse Polish Notation, first the arguments are pushed on a stack and next the commands are given). Next I move to a position near the bottom of the page and select a rotation angle of 90°. The text between the parentheses is printed, I move 40 points down and print the next sentence.

Of course I need to include this header in the PostScript file I send to the printer. Although it is probably possible to do this using TEX specials, I prefer to use `dvips` to do such tasks. The following `dvips`-option suffices: `dvips -h watermark.pro`.

Hans Hagen
pragma@wxs.nl

metapost

Making stand alone METAPOST graphics

keywords
METAPOST, pdf, pdfTeX

abstract
When a METAPOST graphic uses fonts, the PostScript file is not self contained and hardly usable outside TeX. One can however use TeX itself, or actually pdfTeX, to create such a graphic. Although this method uses an ConTeXt module, the solution provided here is independant of this macro package. The macros responsible for the process are collected in the file mptopdf.tex.

The file mptopdf provides a quick way to convert METAPOST files to PDF using a slightly stripped down plain TeX, PDFTeX, and a few CONTEXT modules.

First generate a format, which in WEB2C looks like:

```
pdftex --ini mptopdf
```

Since this conversion only works with PDFTeX or PDF-ε-TeX, the session is aborted when another TeX is used. When finished, the resulting fmt file should be moved to the right location.

The conversion itself is accomplished by:

```
pdftex &mptopdf \relax filename.number
```

The \relax is needed since we don't want to process the file directly. Instead we pick up the filename using \everypar. Since this file is still the first one we load, although delayed, the jobname is as we expect. So, at least in WEB2C, the result of the conversion comes available in the file filename.pdf. This conversion process is roughly compatible with:

```
texexec --pdf --fig=c --result=filename.pdf filename.number
```

This uses CONTEXT, and is therefore slower.

The implementation is rather simple, since we use some generic CONTEXT modules. Because we need a few register allocation macros, we preload plain TeX. We don't load fonts yet.

```
1  \input syst-tex
```

We check for the usage of PDFTeX, and quit if another TeX is used.

```
2  \ifx\pdfoutput\undefined
     \message{Sorry, you should use pdf(e)TeX instead.}
     \expandafter \endinput
   \fi
```

The conversion to PDF is carried out by macros, that are collected in the file:

```
3  \input supp-pdf
```

We use no output routine.

4 **\output***{}*

Since we need to calculate and set the bounding box, we definitely don't want to indent paragraphs.

5 **\parindent**=0pt

We use \everypar to pick up the filename and process the METAPOST graphic.

6 **\everypar***{***\processMPfile***}*

The main macro shows a few PDFTEX primitives. The main work is done by the macro \convertMPtoPDF which is defined in upp-pdf}. Thi macro interprets the METAPOST file. Close reading of this macro will probably learn a few (PDF) tricks. Apart from some path transformations, which are needed since PDF has a different vision on paths, the graphic is positioned in such a way that accuracy in PDF xforms is guaranteed.

```
7 \def\processMPfile#1 %
    {\pdfoutput=1
     \setbox0=\vbox{\convertMPtoPDF{#1}{1}{1}}%
     \ifdim\wd0<1in \message{[warning:  width<1in]}\fi
     \ifdim\ht0<1in \message{[warning: height<1in]}\fi
     \pdfpageheight=\ht0
     \pdfpagewidth=\wd0
     \voffset=-1in
     \hoffset=\voffset
     \box0
     \bye}
```

Since acrobat has troubles with figures smaller than 1 inch, we issue a warning. When embedding graphics in documents, a size less that 1 inch does not harm.

The resulting PDF file is about as efficient as such a self contained file can be. However, if needed, this PDF file can be converted to EPS using for instance the pdftops program (in WEB2C) or GHOSTSCRIPT.

8 **\dump**

Victor Eijkhout

TEXniques

Typesetting CD labels

Introduction

Now that CD burners are becoming standard equipment in personal computers, there is a need for software to typeset CD labels. Of course, one can just squeeze a normal paragraph of text in the confines of a label, but it would be much more elegant to set the text to use all the available space. In this short article I will explain the macros that I wrote during a Christmas holiday, and that contain a few neat tricks.

The full macros can be found on CTAN under

`http://tug.ctan.org/cgi-bin/CTANfilesearch.pl?FILESTRING=cd_labeler`

you will also need

`http://tug.ctan.org/cgi-bin/CTANfilesearch.pl?FILESTRING=repeats`

The plan of attack, and the obvious stumbling block

Since labels are circular, and TEX has a general paragraph mechanism in the `\parshape` primitive, it seems logical to compute line lengths and indentations to get text to fit on a label. In fact, we would need to fit the text in between the large circle of the outer rim, and the small circle of the hole of the label. Basically, from the number of lines that we are below the top or the middle of the CD, plus the `\baselineskip`, we get our y coordinate, and from that we compute the line length and indentation.

Eh, compute exactly how? TEX has only integer arithmetic, and even that is limited to adding, multiplying, and dividing. We need something like a square root to compute $x = \sqrt{r^2 - y^2}$.

In fact, the problems start before the square root computation: I decided to solve the integer problem by using 'scaled points', TEX's smallest measure, of which there are some thousands in a lowly printers point. Now, the radius of a CD label is a little over the square root of the maximum integer representable in TEX; in other words, computing r^2 gives integer overflow. This set me back for a while. I tried scaling down the quantities, taking the square root and scaling them up again, but this is not very elegant. The solution I found was to compute

$$r^2 - y^2 = (r + y)(r - y),$$

which is far less likely to overflow.

Next the problem of computing the square root. Given that TEX does have division the solution is clear: I had to implement Newton's method. (In fact, if TEX hadn't had division I could have implemented that too with Newton's method. In the early days of computing this was how the hardware did things even.) For the uninitiated: Newton's method is an idea of computing the solution of an equation by making successively better approximations. Formally, to compute the value x for which $f(x) = 0$, one computes a series of approximations x_i from

$$x_{i+1} = x_i - f(x_i)/f'(x_i).$$

In the case of square roots, where we want to compute $x = \sqrt{a}$, we let $f(x) = x^2 - a$, giving the iteration

$$x_{i+1} = (x_i + a/x_i)/2.$$

In TEX terms, using the \repeat macro I just explained, this can be written as

```
\count1=#1
\repeat \do {
   \count2=#1\relax \divide\count2 by \count1
   \advance\count2 by \count1 \divide\count2 by 2
   \ifnum\count2<\count1 \count1=\count2
     \else \expandafter\breakrepeat \fi}
```

This is obviously taken from a macro where the first argument is the number we need the root of. (Mathematically inclined TEXnicians may be amused to note the crude but effective stopping test of this iteration.)

User interface

For the user interface I decided on making a box, which actually gets printed, in which all material gets superimposed.

```
\CDLlabel{ <label content> }
```

Available commands for filling the label are:

```
\CDLbackground{
     <picture material> }
\CDLunderhole{<setup>}{
     <material printed below the hole> }
\CDLlowerhalf{<setup>}{
     <material on the lower half of the label,
      left of and below the hole> }
\CDLleft{<setup>}{
     <material printed on the left side of the label> }
\CDLright{<setup>}{
     <material printed on the right side of the label> }
```

Any combination of these can be given; it is up to the user not to give too much text. One thing I didn't implement is filling the whole label with text that would flow around the hole. It can be done, but I have never seen it, so the need must not be overwhelming.

The 'setup' part of the macros is where the \baselineskip setting has to be done; font changes can go there or in the actual text.

The resulting label is in essence a \vbox, so it can be positioned on the paper by the usual shift commands. I found it easiest to shift the \hoffset.

Of the above commands, the 'left' and 'right' are convenient for printing the disc title. However, since they start all the way at the top, the following command may come in handy:

```
\CDLblank{ <number of lines to skip> }
```

Just a little more about the internals

Since any of the above macros can be given in arbitrary combinations, internally they are all put in a box of zero width:

```
\long\def\CDLlabelbox#1{%
     \setbox\CDLbox=\vbox to \CDLdiscsize{\hsize=\CDLdiscsize
         #1\vfil
         \hrule width \CDLdiscsize height 0pt depth 0pt}%
     \wd\CDLbox=0pt \box\CDLbox}
```

The surrounding box is an `\hbox`, so these boxes are all ‘lined up’.

Internally, most of the hard work is done by this macro:

```
%% CDLline
%% from #1 line number compute #2 length, #3 inner gap #4 outer gap
%%
\def\CDLline#1#2#3#4{...]
```

Apart from the first argument, the arguments to this macro are control sequences that get defined in the macro definition. This is then used in such top level macros as

```
\def\CDLleft#1#2{\CDLlabelbox{
   \CDLtoks={}#1\relax
   \begingroup
     \repeat \for{CDLcount} \from{1} \do {%
       \CDLline\CDLcount\a\b\c
       \if\CDLerror \expandafter\breakrepeat \fi
       \edef\temp{\CDLtoks={\c\space\a \space\the\CDLtoks}}\temp
       }%
     \noindent
     \parshape \CDLcount 0pt 0pt \the\CDLtoks \hfill\break #2\par
   \endgroup
   }}
```

Note how the paragraph shape gets built up by gradually adding to a token list.

Conclusion

There are always new challenges for TEX to tackle. So far the primitives seem up to the task: typesetting in a circle was quite easy. Now I should really write a floating point arithmetic package to make the computations a bit more elegant :-)

TEXniques

Victor Eijkhout

The ultimate loop macro

Introduction

The plain TEX format contains a `\loop` macro that has been a source of frustration and puzzlement to users ever since. Its syntax is somewhat strange, you have to insert an `\if...` condition in it but cannot use `\else`, and nested use of the macro runs into various problems. In this article I will describe my own improved loop macro, which I've called `\repeat` to prevent confusion.

You can get the macros from CTAN:

`http://tug.ctan.org/cgi-bin/CTANfilesearch.pl?FILESTRING=repeat`

keywords

...

User interface

Looping constructs have been common in programming languages for a long time. My loop macro is vaguely modelled on the Algol68 construct: the syntax is

```
\repeat
   \for{<var>} \from{<start>} \by{<step>}
               \to{<end>} \downto{<end>}
               \until{<cond>} \while{<cond>}
   \do { <loop body> }
```

Some remarks about this:

- All control sequences in between `\repeat` and `\do` are optional; if you leave them *all* out, you get an infinite loop.
- If a 'for' variable is specified, for instance `\for{i}`, a control sequence `\i` is available in the loop body. Strictly speaking, this control sequence has been `\let` to a counter that is allocated by the package. This loop variable can also be used as a bound for any nested loops.
- The loop body is written inside braces, but there is no implied grouping, so all assignments are global.
- The step size is always positive; it is added or subtracted depending on whether `\to` or `\downto` is used. The default is, of course, an increasing counter, stepping by 1.
- The 'until' test is evaluated at the end of the loop body; the 'while' test at the start. The condition is any TEX `\.` test. To terminate the loop with a test somewhere in the middle of the loop body, use

  ```
  \ifsomething ... \expandafter \breakrepeat \fi
  ```

Implementation

Above, I mentioned the fact that the `\repeat` macro can be used nested; in fact, it can be nested to as many levels as you want. Now, I also mentioned that the loop has a counter. So, do I allocate whole bunch of counters to beging with? Nope. Here's the crucial bit:

```
\newcount\REPdepth
\def\repeat#1\do{%
   \advance\REPdepth by 1
   \REPcsargrom\ifx{REPcount}\relax
     \REPcsargrom{\csname newcount\expandafter\endcsname}{REPcount}%
     \REPcsargrom{\csname newtoks\expandafter\endcsname}{REPtoks}%
```

where

```
\def\REPcsargrom#1#2{%
  \expandafter#1\csname#2\romannumeral\REPdepth\endcsname}
```

That is, a new counter is allocated for each level, the first time it is encountered. A unique token list for each level is also allocated to hold the loop body.

The macro goes on:

```
   \fi \REPsetup{#1}%
   \edef\REPtmp
     {\def\REPcsargrom\noexpand{REPrepeat}%
          {\REPcsargrom\noexpand{REPbody}}}%
   \REPtmp
   \afterassignment\REPdxbody\REPcsrom{REPtoks}}
```

where

```
\def\REPcsrom#1{\csname #1\romannumeral\REPdepth\endcsname}
```

The `\REPsetup` call processes all the options, then the `\edef` trickery defines control sequences such as `\REPrepeatii` (on level 2) as `\REPbodyii`; this superfluous looking step is necessary because we terminate the loop by redefining `\REPrepeatii` as `\relax`. The `\afterassignment` sets aside the 'define and execute' macro `\REPdxbody`, and the token list `\REPtoksii` is then assigned whatever comes after `\do` (remember that the argument of `\repeat` was delimited by `\do`?); in other words, the loop body.

The 'define and execute' macro of the loop body goes like this:

```
\def\REPdxbody{%
   \REPcsargrom\edef{REPbody}{%
     ... % the while test
     \noexpand\the\REPcsargrom\noexpand{REPtoks}%
     ... % the until test
     ... % counter update
     \noexpand\endrepeat
     \REPcsargrom\noexpand{REPrepeat}}%
   \REPcsrom{REPbody}}
```

Above we had defined `\REPrepeatii` as `\REPbodyii`, so together this is a clean case of daisy-chain recursion.

Ending the loop is done by, as promised, by defining away the `\REPrepeatii` control sequence:

```
\let\endrepeat\relax
\def\breakrepeat#1\endrepeat{%
   \REPcsargrom\let{REPrepeat}\relax
   \advance\REPdepth by -1\relax
   }
```

Of course, I have left out plenty of detail here, but this should convey the flavour of these macros.

Examples

If you retrieve the file from CTAN, you'll see various examples at the end, after an `\endinput` statement. Here are a few.

An loop, to be executed three times:

```
\repeat \to{3} \do {
   \message{hello there!}
}
```

Looping until the counter reaches some condition, here divisibility by 37:

```
\newcount\tmpcount
\repeat \for{j}
   \until{\tmpcount\j \divide\tmpcount by 37 \noexpand\ifnum\tmpcount=1}
  \do {
    \message{testing \number\j}
}
```

An example of nested loops, where the inner loop uses the loop counter of the outer loop in its bounds:

```
\repeat \for{i} \by{2} \to{10} \do
  {\repeat \for{j} \from{\i} \by{3} \to{18} \do
   {\message{(\number\i.\number\j)}
  }}
```

That's it, folks. I have a hard time imagining that someone could want yet more from a loop macro, but if you can think of something, just let me know.

Hans Hagen
pragma@wxs.nl

ConTEXt en pdf
Postprocessing pdf **files**
an application of TEXexec **and** pdfTEX

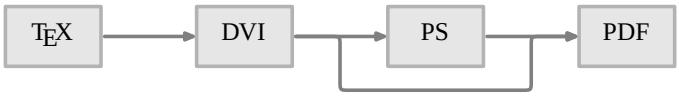
keywords
pdf, postprocessing, TEXexec, pdfTEX

abstract
This article introduces some ways to manipulate pdf files using pdfTEX, ConTEXt, and TEXexec. The method described here can be used for arbitrary pdf input, given that it can be handled by pdfTEX.

1 Introduction

The traditional TEX workflow can be summarized as follows:

A slight variation to this workflow is the direct conversion of DVI into PDF:



Both flows share that the intermediate formats DVI and POSTSCRIPT can be postprocessed with utilities, for instance to produce A5 booklets from A4 documents. Since TEX macro packages can use the `\special` primitive to add directives to the DVI file, effects not directly supported by TEX the program, can be achieved.

The previous chart shows a more direct way to produce PDF, the flow supported by PDFTEX. It will be clear that postprocessing now must take place at the PDF level.

Since PDFTEX can include pages from PDF files in a document, postprocessing can be handled by itself. In other words: PDFTEX can manipulate PDFTEX output. An advantage of this approach is that fonts are embedded efficiently. However, far more important is that one can use TEX to enhance the original documents while processing them again.

When postprocessing a PDF file, we can distinguish two categories: page imposition, which may lead to reordening of the pages, and collecting, which has a more linear nature.

I will limit the descriptions to the functionality as provided by TEXEXEC, the com-

mand line interface to CONTEXT. This does not mean that postprocessing is limited to files produced by CONTEXT: any reasonable and valid PDF file can be handled. TEXEXEC is only a PERL based wrapper, that generates the appropriate (relatively small) TEX files that do the job. By looking at the generated files `texexec.tex` one can get some insight in the CONTEXT commands involved.

2 Combinations

Especially presentations can be chacterized by an inefficient use of paper: relatively large fonts are used and the amount of text on a page is rather minimal. Therefore, when we want to print them, it makes sense to combine many of those pages on one sheet of paper. Such a page can be generated by saying:

```
\combinepages[pre-symb][nx=3,ny=5]
```

Of course it is not that convenient to key in commands like this for simple jobs, although a manual setup has the benefit that we can set more parameters than shown here. Using the default settings, TEXEXEC provides:

```
texexec --pdfcombine --combination=3*5 pre-symb
```

The small pages will be scaled in such a way that they comfortably fill the page. This is demonstrated on the next page. There are a few switches that controll the output:

`paperformat`	a predefined CONTEXT paper size, like `letter` or `A4`
`paperoffset`	a dimension specifying the margins in TEX units
`combination`	a `n*m` grid limited by the number of pages

3 Copying

Some printers, like ink–jet printers, have a relatively large unprintable area. The next command scales down a file so that it fits comfortably on the paper.

```
texexec --pdfcopy --scale=.95 yourfile.pdf
```

When one knows the unprintable margins, providing an offset makes more sense. The next call makes CONTEXT calculate the scale automatically:

```
texexec --pdfcopy --paperoffset=1.5cm yourfile.pdf
```

Both calls are especially useful when for instance the title page uses graphics (or color) that runs off the page.

`scale`	a (floating point) number like 0.85
`paperoffset`	a dimension specifying the margins in TEX units

4 Arranging

Say that one does not want to spend paper on printing the PDFTEX manual. Instead of printing he manual on A4, one can produce an A5 booklet.

```
texexec --pdfarrange --paper=a5a4 --print=up pdftex-a
```

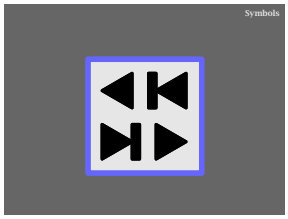
In this case it makes sense to add the following switch:

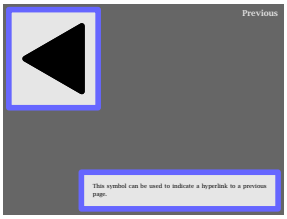
```
--addempty=1,2
```

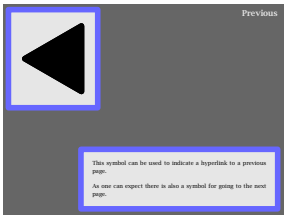
This directive tells CONTEXT to add two empty pages after page 1 and 2 (the title pages).

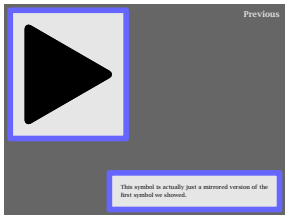
When asking for help (`--help pdfarrange`) one gets a list op options.

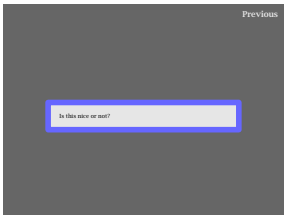
`paperoffset`	a dimension specifying the margins in TEX units
`paper`	a mapping like `a5a4` or `a4a3`

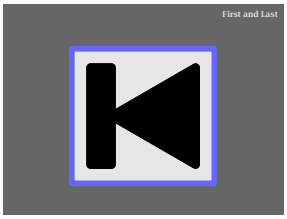

Symbols

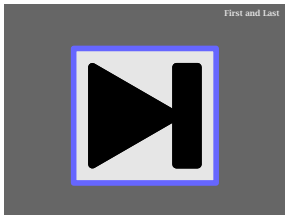

Previous
This symbol can be used to indicate a hyperlink to a previous page.

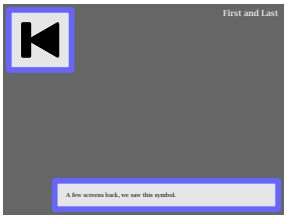

Previous
This symbol can be used to indicate a hyperlink to a previous page.
As one can expect there is also a symbol for going to the next page.

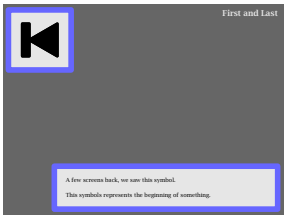

Previous
This symbol is actually just a mirrored version of the first symbol we showed.

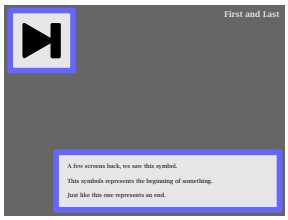

Previous
Is this nice or not?

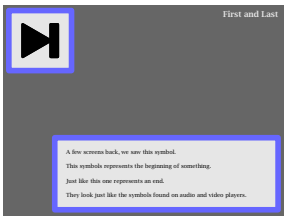

First and Last

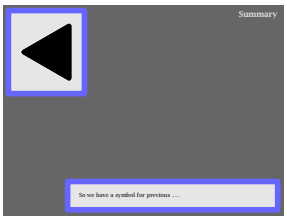

First and Last

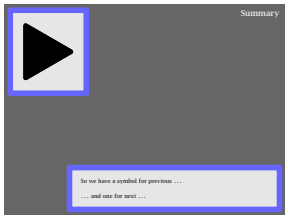

First and Last
A few screens back, we saw this symbol.

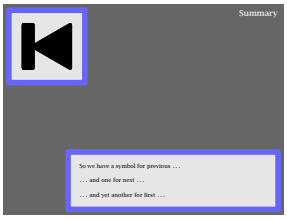

First and Last
A few screens back, we saw this symbol.
This symbols represents the beginning of something.

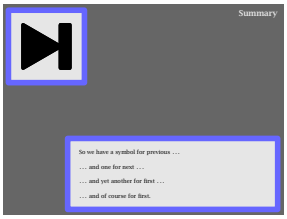

First and Last
A few screens back, we saw this symbol.
This symbols represents the beginning of something.
Just like this one represents an end.


First and Last
A few screens back, we saw this symbol.
This symbols represents the beginning of something.
Just like this one represents an end.
They look just like the symbols found on audio and video players.


Summary
So we have a symbol for previous ...


Summary
So we have a symbol for previous ...
... and one for next ...


Summary
So we have a symbol for previous ...
... and one for next ...
... and yet another for first ...


Summary
So we have a symbol for previous ...
... and one for next ...
... and yet another for first ...
... and of course for first.

`print`	an arrangement like `up` or `down`
`noduplex`	when issued, it forces single sided output
`backspace`	the side (inner) margin of the page in TEX units
`topspace`	the top (and bottom) margin of the page in TEX units
`markings`	when issued, cutmarks are added
`addempty`	a comma delimited list of pages after which to add an empty page
`textwidth`	the width of the original in TEX units (single sided)

In case of a single sided original with an asymetric layout, the width of the text should be specified to get the best results.

By providing more than one filename, one can combine files. This enables the user to add for instance a title and/or colofon page.

5 Selecting

One can create a stripped down version of a document by using `--pdfselect`. The next example filters some pages from two presentations and combines them into one document.

```
texexec --pdfselect  --paper=S6 --selection=1,3,8 --result=r-1 p-1
texexec --pdfselect  --paper=S6 --selection=2,5,9 --result=r-2 p-2
```

We can follow this up by:

```
texexec --pdfarrange --paper=S6 --noduplex --result=p-3 r-1 r-2
```

Again, there are some options:

`selection`	a list of pages to select, `odd` or `even`
`paperoffset`	a dimension specifying the argins in TEX units
`paperformat`	a predefined CONTEXT paper size, like `letter` or `A4`
`noduplex`	when issued, results in single sided output
`backspace`	the inner margin of the page in TEX units
`topspace`	the top margin of the page in TEX units
`markings`	when issued, add cutmarks
`addempty`	a list of pages after which to add an empty page
`textwidth`	the width of the original (one sided case)

6 Remarks

As I already pointed out, TEXEXEC's main task is to provide a proper command line interface to CONTEXT. Options are written to a option file, TEX is called with the CONTEXT format, and CONTEXT reads the options. When the job is finished, TEXEXEC calls TEXUTIL to sort the index, and, if needed, takes care of additional passes. Without changing the source file, one can invoke specific environments and style options, called modes.

Since its main task is to manage TEX runs, TEXEXEC can also be used to generate overviews of graphics, make listings of source code, generate module documentation, prepare formats, etc.

In the perspective of postprocessing PDF files the following option is worth mentioning:

```
texexec ...... --result=filename
```

By default, the results go to the file known as `\jobname`, which in the case of postprocessing PDF is `texexec.pdf`. The `--result` switch can be used to specify an alternative output file.

Another usefull option is `--help`, that can be followed by a switch specifier to get more help.

```
texexec --help pdfarrange
```

When using CONTEXT as macro package for processing TEX files, instead of arranging PDF pages, one can also rely on the built in page imposition mechanisms. These cover a rather wide range of possibilities and can be set up in the main document style. The `--noarrange` and `--arrange` switches control this process.

As demonstrated in a previous section, page imposition without the need to add directives to the document style is also possible. While the `--arrange` switch typesets the document at the requested size, the `--pdfarrange` option simply scales the pages and arranges them as images. Therefore:

```
texexec --arrange --paper=a5a4 --print=up somefile
```

and

```
texexec --pdfarrange --paper=a5a4 --print=up somefile
```

are fundamental different operations. The first one involves typesetting and moving pages around, the second concerns copying, scaling and moving of already typeset pages.

More information on these and other options can be found in the TEXEXEC manual. We expect to add more postprocessing features and options in the future. For more advanced and complicated cases one can always define a dedicated CONTEXT source file.

Since these facilities are still being extended and optimized, it makes sense to use the latest versions of PDFTEX, CONTEXT, and TEXEXEC. More information can be found at our home page: `www.pragma-ade.nl`, CTAN or one of the CONTEXTmirrors.

Hans Hagen
pragma@wxs.nl

ConTEXt en pdf
Annotating presentations

keywords
ConTEXt, presentations

abstract
Today most presentations are enlightened by text shown on transparancies or using video beamers. This text is often rather limited in size. In this article I present a method of annotating pages that can be used with the ConTEXt presentation styles.

Currently there are some 16 presentation styles available in CONTEXT. The first 6 are already public, the next 10 will follow soon, and more are in preparation. The document source of a presentation using the 16th style may look like:

```
\usemodule[pre-16]

\starttext

\StartIdea

\Topic{Ideas}

The first series of presentation styles was rather
straightforward in design and provided mere alternatives
for traditional sequential presentations.

\NextIdea

Later styles are different and even a bit weird. They sort
of reflect a presentation style and guide the speaker through
his or her presentation.

\StopIdea

\stoptext
```

The text entered here shows up in the final product. However, instead of using such verbose phrases, one can for instance present graphics, show snippets of code or summarize conclusions using itemized lists. In these situations the author may want to annotate the pages in such a way that he or she will have some guidance during the talk.

For that purpose, CONTEXT provides a general mechanism for annotating pages. Since these annotations are typeset on a separate layer of the printing paper area, they don't interfere with the layout of the document.

The annotations, called page comment, should be entered before the page is flushed. Therefore, in our example they should be entered before `\StopIdea`.

```
\startpagecomment

We can add comment to a presentation. This comment is hidden
for the audience. However, the speaker can generate a
print||out with the comments nicely typeset.

\stoppagecomment
```

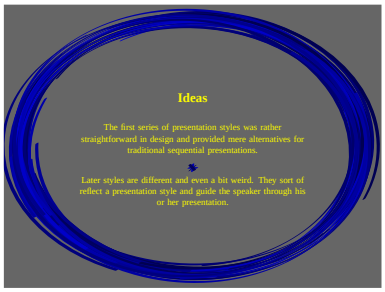
Annotations don't show up unless their they are enabled, for instance by saying:

\setuppagecomment[state=start,location=right]

By default the state=stop. Alternative locations are bottom, left, and top. The annotations can be positioned in more detail by setting the next parameters:

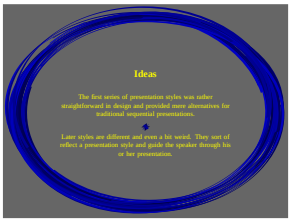
keyword	meaning	default
offset	the gap around the page and annotation	0.5cm
distance	the gap between the content and annotation	0.5cm
width	the width of the annotation text	10.0cm
height	the height of the annotation text	5.0cm

We show the four alternatives using the the previously defined, one page, presentation.



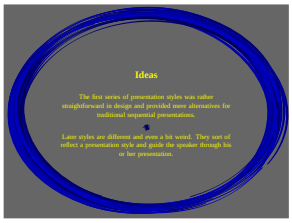
We can add comment to a presentation. This comment is hidden for the audience. However, the speaker can generate a print–out with the comments nicely typeset.

location=right



We can add comment to a presentation. This comment is hidden for the audience. However, the speaker can generate a print–out with the comments nicely typeset.

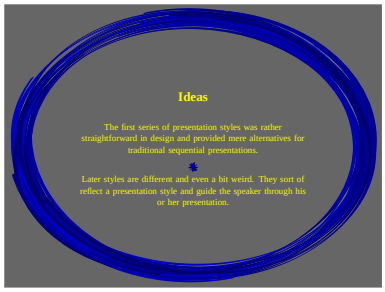
location=top



We can add comment to a presentation. This comment is hidden for the audience. However, the speaker can generate a print–out with the comments nicely typeset.

location=bottom

We can add comment to a presentation. This comment is hidden for the audience. However, the speaker can generate a print–out with the comments nicely typeset.



location=left

Instead of explicitly turning the annotations on, one can provide a run–time option to TEXEXEC (a PERL script that provides a command line interface to TEX), like:

```
texexec --mode=comment --pdf somefile
```

Of course one can also enable this style mode in the document source itself, saying:

\enablemode[comment]

This mode is implemented in the base presentation style that is loaded by default.

tutorial

Literate Programming

Michael A. Guravage
NLR
Anthony Fokkerweg 2
1059 CM Amsterdam NL
guravage@nlr.nl

abstract

This article is a short introduction to the theory and practice of a programming style known as Literate Programming; a style that changes the focus of writing programs away from telling a computer what to do and toward explaining to a person what it is we are telling the computer to do. Literate Programming overcomes the limitations inherent in presenting traditionally structured program text. Using a balanced mix of informal and formal methods, literate programs are presented in a way suited for human understanding. Processing a literate program source results in both a nicely typeset document describing the parts of the program in an order that elucidates their design, and source code in an order in which it will compile.

keywords

Literate Programming, Structured Programming, WEB

Introduction

Writing good programs is hard; so is writing good program documentation.

Structured programming in the 1970's and Object Oriented Technologies in the 1980's are methods that help us capture and organize the design and implementation of complex software. However, though our design decisions are embodied in the code we write, they are often obscured by the code itself. Few people enjoy navigating through pages of curly-braces, control-structures, and function calls in unfamiliar code. Literate Programming provides a way to expose and elucidate a program's design by presenting its parts in an order and at a level of abstraction that places a premium on understanding.

The tasks of writing a program and writing program documentation are often seen as separate and sequential. There are several undesirable consequences of this separation. First is that their writing does not inform each other. If the program is written before its documentation, the program's relationship to its documentation is merely coincidental. Second is that separate code and documentation tend to diverge over time. As a consequence, the code and documentation do not cooperate as well as they could toward either maintenance or reuse. Literate Programming tries to address these issues by integrating, or if you prefer - blurring the distinction between, code and documentation in such a way that code and documentation contribute to and complement each other.

The remainder of this article will be arranged as follows: we begin by discussing the motives for and ideas behind literate programming. Next, we identify the properties that characterize literate programs. The process of transforming literate programs into running code and typeset documents is explained. We compare language dependent and language independent literate programming tools and enumerate the benefits and liabilities of each. Lastly, the costs of using literate programming are estimated.

Motivation for Literate Programming

Literate Programming originated in the early 1980's as part of Knuth's work on TEX and digital typography. By this time, the concept of *structured programming* was already well established. Structured programming advocates decomposing a problem into a hierarchy of smaller problems according to some subordinating principle. The level of abstraction proceeds from generalities to specifics until each task at the bottom of the hierarchy admits a solution. One useful criteria for subdividing tasks into smaller ones is that it should be possible at each level of abstraction to give an informal description of its subtasks.

Marc van Leeuwen (van Leeuwen, 1990) observed that, "Although the composition of a structured program should reflect the design decisions that led to its construction, the traditional way of presenting such programs, e.g. listings of code containing comments, lacks the appropriate facilities for communicating this information effectively to the readers of the program, seriously limiting the readability, especially to people other than the programmer of the code. Yet readability is of vital importance, because it is only by careful reading that we can verify that the design of a program is sound and well-implemented, and to understand where and how changes can be made when such a need arises."

Knuth expressed what many have long recognized, that the order in which a traditional program is written is a concession to the computer. A Euclidean proof begins with first principles, and builds a consistent hierarchy of definitions, postulates, and theorems in service of a proof. Likewise, a computer program is written as a hierarchy

of definitions, function prototypes, data structures, and instructions whose order satisfies the demands of a computer. While the Euclidean style of proof is unparalled for its completeness, its style is less applicable when the goal is to convey a general appreciation for a subject to the uninitiated. Though a computer program implementing an algorithm is sufficient to instruct a computer how to perform the algorithm, a listing of the program may be, and often is, insufficient to explain the workings of the algorithm to a person. Vice versa, an informal description of an algorithm can reveal to a person how the algorithm works, the same description is useless to a computer. It is apparent that there is one way a program must be ordered in order to be parsed and compiled by a computer, but there may be many different ways to arrange the program to explain its workings to a person.

"Literate Programming is a natural sequel to structured programming (van Leeuwen, 1990)", and overcomes the shortcomings of the latter by combining informal natural language, a formal programming language, and a flexible order of elaboration in such a way that places a premium on exposition and understanding. By tightly coupling program code and program documentation, a literate programmer strives to write programs that are comprehensible because their concepts have been introduced in an order and at a level of detail that is suited for human understanding (Knuth, 1992a).

What's in a name?

The origin of the term *literate programming* has both a serious and a light side. On a serious side, Knuth found that the more he concentrated on developing a programming style that concentrated on clarity of exposition, the more he treated his programs as works of literature, the better his programs became. Their designs were improved, their implementations had less bugs than their traditional counterparts; and they were readable. Like an author writing a story or a researcher writing a technical article, the programmer adopts the common goal of tell his audience just what they need to know, just when they need to know it.

On the lighter side, Knuth wrote, "I must confess that there may also be a bit of malice in my choice of a title. During the 1970s I was coerced like everyone else into adopting the ideas of structured programming, because I couldn't bear to be found guilty of writing unstructured programs. Now I have a chance to get even. By coining the phrase literate programming, I am imposing a moral commitment on everyone who hears the term; surely nobody wants to admit to writing an illiterate program (Knuth, 1992a)."

Knuth also coined the term WEB to describe a literate program. He thought that a complex piece of software is best regarded as a web that has been delicately pieced together from simple materials (Knuth, 1992a).

To complete the literate programming nomenclature, TANGLE is the name of the processes that rearranges a literate program in an order that is easier for a computer to understand; while WEAVE is the name of the process that rearranges a literate program into an order that is easier for a person to understand. Throughout this article, the names WEB, TANGLE, and WEAVE will refere to these processes; without reference to specific literate programming tools with the same names.

Properties of Literate Programs

A program has to exhibit three properties before it can be called a literate program. The first property is that a single literate program source should, when processed, produces both a runnable program and a nicely typeset document. The work of *tangling* the code and *weaving* the document will be explained in detail in the following sections. The second property is that a literate program must exhibit a flexible order of presentation. As has already been mentioned, the order of presentation to a person is independent of the order of compilation by a machine. The third and last property is that a literate program, and the tools that process it, should facilitate the automatic generation of cross-references, indices, and a table of contents.

Anatomy of Literate Programs

Knuth (Knuth, 1992a) describes a literate program as, "A traditional computer program that has been cut up into pieces and rearranged into an order that is easier for a person to understand. A traditional computer program is a literate program that has been rearranged in an order that is easier for a computer to understand. Literate and traditional programs are essentially the same kind of things, but their parts are arranged differently. You should be able to understand the traditional program better in its literate form, if its author has chosen a good order of presentation."

A literate program consists of numbered *sections* or *chunks*. A section can be either named or unnamed and corresponds to a paragraph in written language. Within a section, a single idea is developed. A section is divided into two parts: an informal specification in a natural language, and a formal specification in a programming language. The informal half contains a written description of the idea which is the focus of the section. The formal half contains the code implementing that idea.

To earn the title of literate programmer, one needs to add a handful new control sequences to his repertoire of programming tools. Most begin with an '`@`' and control such things as sectioning, cross-references, and layout. Figure

2 shows a portion of literate C source code extracted from the word count example program that comes with the CWEB literate programming package. The section begins with an '`@`' followed by several lines of commentary. Next is the section name which appears between angle brackets: '`@< Name of section @>=`.' The section name delimits the informal commentary from the formal code - 'also called the *replacement text*.'

In this example the code is dominated by a '`do ... while`' loop; which contains several references to other sections whose names each appear between '`@<`' and '`@>`' pairs. Notice that, at this level of abstraction, the program text for the inner levels are suppressed, making the outline of the outer level more clear.

Processing a WEB

A literate programmer writes code that serves as the source for two different system routines. Figure 1 shows the two paths a literate program can take. One path is called *weaving* the web; the result of applying WEAVE and TEX is a nicely typeset document. The other path is called *tangling* the web; the result of applying TANGLE is program code rearranged in an order ready for compilation.

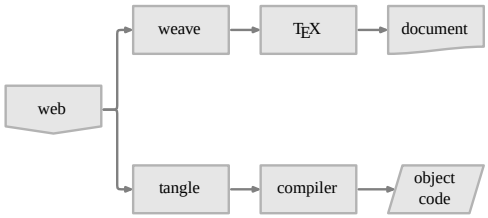


Figure 1 Processing a WEB

WEAVE

Figure 3 shows the result of passing the previous piece of literate code through WEAVE and then typesetting it with TEX. To enhance readability, reserved words, like '**do** and **while**', are set in boldface type, and identifiers, like '*argc*', are set in italics.

Note that WEAVE has resolved the section numbers, references, and cross-references. The footnote at the end of the section complements the section number by showing in which other sections the current section is referenced.

TANGLE

TANGLE removes all the commentary from a WEB and rearranges the code into an order in which it will be compiled - see Figure 4. The reordering porceeds as follows: first, all unnamed sections are collected in relative order; then references to named sections are replaced with their replacement text. This continues until all section references have been replaced.

Like object code, the tangled source code can be considered incidental. However, to help you navigate through the code if you venture to read it, TANGLE adds comments that show which literate section the code originated. TANGLE also can add '`#line`' directives to allow compilers and debuggers to map lines in the tangled code back to lines in the original literate source.

A comparison of the tangled code to either the original literate source or the woven result should leave no one in doubt how TANGLE got its name.

Tools for Literate Programming

Literate Programming tools can be divided into two broad groups: those that are language dependent and those that are language independent. Members of the first group include WEB - Knuth's original literate programming environment for Pascal, CWEB - an adaption of WEB to C by Silvio Levy, and CWEBX - an implementation of CWEB by Marc van Leeuwen.

Language dependent tools tend to be large monolithic programs. This makes them easier to port but difficult to modify or extend. Knowing the syntax of the language for which they are written, language dependent tools provide native support for pretty-printing. They also recognize language specific types, which facilitates the automatic indexing of function and variable names.

An example of a language independent literate programming tool is NOWEB written by Norman Ramsey(Ramsey, 1993). The motivation behind NOWEB was that its author thought WEB and its derivatives were too complex and inflexible. The result was a literate programming tool that is simple, extensible, and independent of the target programming language. Restricting itself to writing named chunks of code in any order, with interleaved documentation, NOWEB provides much of the functionality of WEB at a fraction of the complexity.

Unlike monolithic language dependent tools, NOWEB adopts a pipelined architecture. Like the UNIX notion of pipes where the output of one program is the input to another, the transformation from literate source to typeset document, or program code, is achieved via a succession of distinct steps. Besides NOWEB's predefined pipeline, filters can be added to both NOWEAVE and NOTANGLE allowing one to easily change their behavior or add new features. Filters add such features as pretty-printing, indices, and cross-references. Currently NOWEB produces TEX, LATEX, HTML, and TFOFF.

```
@ Now we scan the remaining arguments and try to open a file, if
possible.  The file is processed and its statistics are given.
We use a |do|~\dots~|while| loop because we should read from the
standard input if no file name is given.
@<Process...@>=
argc--;
do@+{
  @<If a file is given, try to open |*(++argv)|; |continue| if unsuccessful@>;
  @<Initialize pointers and counters@>;
  @<Scan file@>;
  @<Write statistics for file@>;
  @<Close file@>;
  @<Update grand totals@>; /* even if there is only one file */
}@+while (--argc>0);
```

Figure 2 A portion of literate source code

Counting the Cost

At the lowest level, where we consider machine time, the additional time needed to process a literate program is negligible. With all the mechanics burried in a makefile, invoking TANGLE takes just slightly longer than the time it takes to compile the resulting code. Likewise, invoking WEAVE does not take long to produce a file, but typesetting the result with TEX does take longer by comparison. Still, typesetting even a moderate size document goes rather quickly.

On a higher level there is no consensus on how long it takes to write a literate program in comparison to writing a traditional program. Knuth says that the time he needs to write and debug a literate program is no longer than that for a traditional program. He contends that any extra time he spends writing commentary is recovered because the result needs less debugging. The best one can say is that the cost of creating a high-quality, well documented literate program is the same order of magnitude as writing and documenting an equivalent traditional program.

Literate Programming scales well and can be applied to everything from one-off disposable examples to complicated programming projects. The quality of the end result is a function of how much time and effort you are willing to expend to achieve it.

Style

What style of writing is best suited to literate programming? Since literate programming grants the programmer the same freedom of expression as any author, there is no definitive style. But there are a few useful guidelines to follow. Choose section names that are long enough to capture the meaning of the code in that section. Another tip can be found in the word count program (Figure 3) where Knuth strengthens the section names by begining each with an imperative verb. Search for a balance between formal and informal exposition so you can convey the essence of a section clearly, without distracting the reader with unnecessary detail.

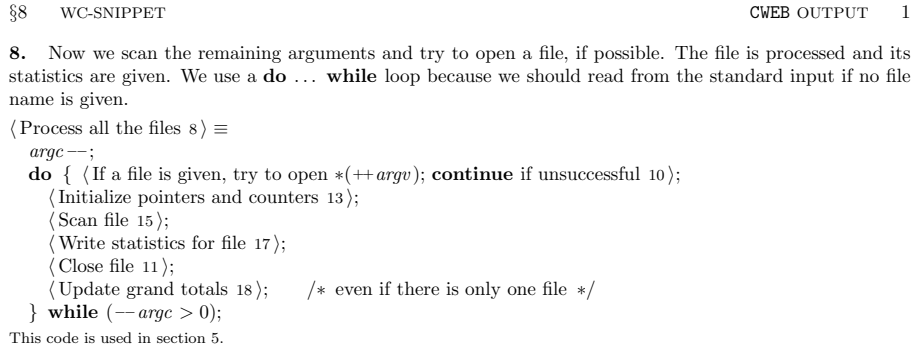
For general remarks on writing styles, you will find some excellent advice in George Orwell's 1946 essay with the foreboding title: "Politics and the English Language(Orwell, 1946)." He encourages the scrupulous writer, in every sentence that he writes, to ask the following questions, "What am I trying to say? What words will express it? What image or idiom will make it clearer? Is this image fresh enough to have an effect? Could I put it more shortly?"

Another indispensable guide to writing is Strunk and White's classic "Elements of Style(Strunk and White, 1979)" where Strunk advises us to be, "direct, simple, brief, vigorous, and lucid."

Summary

The main points to remember when thinking about literate programming are:

- Literate programs use a balanced mix of formal and informal methods, and a flexible order of elaboration, to present information in an order and at a level of detail suited for human understanding.
- Literate Programming does not enforce any one programming paradigm.
- A single literate program produces both a runnable program and a nicely typeset document.
- Literate Programming tools can be divided into two broad groups: those that are language dependent and those that are language independent.
- The cost of writing a literate program is usually equal to or greater than that of writing a traditional one,



§8 WC-SNIPPET CWEB OUTPUT 1

8. Now we scan the remaining arguments and try to open a file, if possible. The file is processed and its statistics are given. We use a **do** ... **while** loop because we should read from the standard input if no file name is given.

⟨ Process all the files 8 ⟩ ≡
  argc−−;
  do { ⟨ If a file is given, try to open ∗(++*argv*); **continue** if unsuccessful 10 ⟩;
    ⟨ Initialize pointers and counters 13 ⟩;
    ⟨ Scan file 15 ⟩;
    ⟨ Write statistics for file 17 ⟩;
    ⟨ Close file 11 ⟩;
    ⟨ Update grand totals 18 ⟩; /∗ even if there is only one file ∗/
  } **while** (−−*argc* > 0);
This code is used in section 5.

Figure 3 The result of applying WEAVE

```
#line 106 "wc.w"
argc--;
do{/*10:*/
#line 127 "wc.w"
if(file_count>0&&(fd=open(*(++argv),READ_ONLY))<0){
fprintf(stderr,"%s: cannot open file %s\n",prog_name,*argv);
status|=cannot_open_file;
file_count--;
continue;
}/*:10*/
#line 108 "wc.w"
;/*13:*/
#line 154 "wc.w"
ptr=buf_end=buffer;line_count=word_count=char_count=0;in_word=0;/*:13*/
#line 109 "wc.w"
;/*15:*/
```

Figure 4 The result of applying TANGLE

but time is often regained since literate programs take less time to debug.

Resources

If you are interested in learning more about literate programming, you should begin with Knuth's book, "Literate Programming(Knuth, 1992a)." Manuals for CWEB and CWEBX can be found in their respective distributions. In addition, the manual for CWEB is published as a book entitled: "The CWEB System of Structured Documentation(Knuth and Levy, 1994)."

Volumes A and B of Knuth's series on Computers & Typesetting describe TEX and METAFONT; volumes C and D contain their literate Pascal sources(Knuth, 1993a, 1992b). A fine example of programming with CWEB is "The Stanford GraphBase(Knuth, 1993b)", Knuth's book on combinatorial data structures and programs.

There are several online resources: the literate programming frequently asked questions can be found at: `shelob.ce.ttu.edu/daves/lpfaq/faq.html`. The ftp site for the literate programming archive is: `ftp.th-darmstadt.de`, and the literate programming newsgroup's name is: `comp.programming.literate`.

A Trustworthy Opinion

Knuth's own opinion about literate programming is expressed in this little WEB which appears on the cover of his book of the same name.

```
⟨ Emphatic declarations 1 ⟩;
  examples: array [vast] of small .. large; beauty; real;
⟨ True confessions 10 ⟩;
  for readers(human) do write(webs);
  while programming = art do
    begin incr(pleasure); decr(bugs); incr(portability);
    incr(maintainability); incr(quality); incr(salary);
    end   { happily ever after }
```

This code is used in theory and practice.

References

van Leeuwen, M. A. A. (1990). *Literate Programming in C*. Manual for CWEBx.

Knuth, D. E. (1992a). *Literate Programming*. CLSI. Lecture Notes Number 27.

Ramsey, N. (1993). Literate-programming tools can be simple and extensible. .

Orwell, G. (1946). Politics and the english language. essay.

Strunk, W. and White, E. B. (1979). *The Elements of Style*. Macmillian, 3rd edition.

Knuth, D. E. and Levy, S. (1994). *The* CWEB *System of Structured Documentation*. version 3.0.

Knuth, D. E. (1993a). *TEX the program*, volume B of *Computers & Typeseting*. Addison Wesley.

Knuth, D. E. (1992b). METAFONT *the program*, volume D of *Computers & Typeseting*. Addison Wesley.

Knuth, D. E. (1993b). *The Stanford GraphBase*. ACM Press.

tutorial
LaTeX in proper ConTeXt

Berend de Boer
berend@pobox.com

1 ConTeXt for LaTeX users

1.1 For who is this document

If you are a LaTeX user, switching to an entirely different macro–package is a very big step. Everything you put so much effort in to learn, doesn't work anymore.

I can't help but quote from that famous book "The psychology of computer programming" (chapter 11):

> In making our adjustments to our particular programming language, we can easily become attached to it simply because we now have to much invested in it. We often listen to a man complaining about his nagging, slovenly, and prodigal wife, only to find that when asked why he doesn't leave her, he replies that he cannot live without her. Most people would prefer almost any amount of pain to giving up the familiarity of some constant companion for an unknown quantity. We see this effect when we try to teach a programmer his *second* language. Teaching the first is no great problem, for he has no investment in any other. By the time he has learned two or more, he is aware that more things exist in this world than he has dreamed of. But letting go of the first is, to him, just a promise of pain with no promise of compensating pleasure.

To help lessen the pain for users make the switch, this document shows short LaTeX code snippets and how you do the same in ConTeXt.

ConTeXt is a macro package that's far more advanced than LaTeX. You can enhance LaTeX with third party packages, but not all macro packages work together with each other. ConTeXt is an integrated, powerful and flexible macro package for which you seldom need third party packages. ConTeXt also has been used to create large and complex on–screen documents, including hyperlinks, on screen buttons, forms, cross–document links, and so on.

This document is not a reference to the ConTeXt manual. It only shows you the ConTeXt macros for the familiar LaTeX macros. It does not explain the ConTeXt

macros in detail nor shows you the numerous options almost every command has. You are referred to the http://www.pragma-ade.nl/general/manuals/beta/cont-nli.pdf instead.

1.2 References

If you take up ConTEXt, you probably need help. You can find it at the following locations:

- The ConTEXt support site is http://www.pragma-ade.nl.
- There is a beginner's manual in english at http://www.pragma-ade.nl/zipped/bman-en.zip.
- ConTEXt main reference however is still in dutch and can be found at http://www.pragma-ade.nl/general/manuals/beta/cont-nli.pdf.
- The 4TEX manual also contains much material on ConTEXt, see http://4tex.ntg.nl/4tex5.
- There is a ConTEXt mailling list. Subscribe by sending a message to mailto:majordomo@ntg.nl. In the body of the message type:

```
subscribe ntg-context yourname@yourserver.yourdomain
```

1.3 Acknowledgements

Thanks to David Arnold, Wybo Dekker and Hans Hagen for offering suggestions and sending corrections to this document.

2 ConTEXt basics

This chapter tells you ConTEXt's basics.

2.1 A basic document

A basic 'hello world' kind of document in LATEX looks like:

```
\documentclass{article}

\begin{document}
Hello world.
\end{document}
```

In ConTEXt it looks like:

```
\starttext
Hello world.
\stoptext
```

ConTEXt does not have a special command to load the style for a certain document. If you have them, you can load style files and such with the TEX macro \input.

2.2 Compiling your document

If your LATEX 'Hello world' document is called text.tex, you probably compile your document with:

```
latex test.tex
```

In ConTEXt you use a wrapper around the TEX compiler to compile your document. Your ConTEXt document is compiled with:

```
texexec test.tex
```

texexec also takes care of the table of contents, indexes, references and sorted lists (see also section 11.1). It recompiles your document as many times as necessary to make sure references in the document are ok.

While you could use something as:

```
context test.tex
```

to compile your document, this is not advised. Learn to use `texexec`:

```
texexec test.tex
```

The first output of `texexec` should look like:

```
TeXExec 1.9 - ConTeXt / PRAGMA ADE 1997-2000

           executable : pdfetex
               format : cont-en
            inputfile : test
               output : dvips
            interface : en
         current mode : all
              TeX run : 1
```

and then it starts compiling your document.

ConTeXt supports various language specific interfaces. This document presents the english language interface but ConTeXt also has a dutch and german based interface. If the default interface on your system is not correct, you can either change the initialization file `texexec.ini` or give a parameter to `texexec`:

```
texexec -interface=en test.tex
```

ConTeXt supports also various output formats like `.dvi` and `.pdf`. To generate a `.pdf` document instead of a `.dvi` document, change `texexec.ini` or call `texexec` with the following parameter:

```
texexec -output=pdftex test.tex
```

Or even shorter with:

```
texexec --pdf test.tex
```

Probably the best option is to put the `texexec` parameters as the first line in your document like:

```
% interface=en output=pdftex

\starttext
This example compiles with the english interface and
generates pdf output.
\stoptext
```

2.3 Chapters and sections

Chapters and sections in ConTeXt are very much like LaTeX. In LaTeX you write:

```
\documentclass{report}

\begin{document}

\chapter{One}
My first chapter.

\chapter{Two}
My second chapter.

\end{document}
```

In ConTEXt this looks like:

```
\starttext

\chapter{One}
My first chapter.

\chapter{Two}
My second chapter.

\stoptext
```

Not much difference here. However, referencing to chapters or sections is done differently in ConTEXt, see section 8.1.

2.4 Table of contents

A table of contents in LATEX is done with the `\tableofcontents` command:

```
\documentclass{report}

\begin{document}

\tableofcontents

\chapter{One}
My first chapter.

\chapter{TwoOne}
My second chapter.

\end{document}
```

In ConTEXt you get a table of contents with the `\completecontent` command:

```
\starttext

\completecontent

\chapter{One}
My first chapter.

\chapter{Two}
My second chapter.

\stoptext
```

Remember to compile with `texexec`, or else you will not get the table of contents (see section 11.1). Besides `\completecontent`, ConTEXt also has `\placecontent`. `\completecontent` starts a new page, while `\placecontent` doesn't.

3 Common LATEX Environments

In the following sections the most common LATEX environments are discussed and ConTEXt alternatives are given. Environments are discussed in alphabetic order.

3.1 The abstract environment

With LATEX's abstract environment you can mark one or more paragraphs in some special way. It also depends on the documentclass you have loaded, either article or report. Its layout is equal to the quotation environment.

You use abstract as follows:

```
\documentclass{article}
\begin{document}

\begin{abstract}
This is an abstract of this article.
\end{abstract}

\end{document}
```

There is no abstract environment in ConTeXt, because it really is a style option. However, you can get the same effect as LaTeX's article abstract environment with:

```
\starttext

\startnarrower\switchtobodyfont[small]
\midaligned{\bf Abstract}\par
This is an abstract of this article.
\stopnarrower

\stoptext
```

With `\startnarrower` you get a paragraph, left and right indented by some white space. With `\switchtobodyfont` you get a font somewhat smaller than the current body font. With `\midaligned` you get a centered line.

3.2 The bibliography environment

In LaTeX you can create a bibliography within the `bibliography` environment environment:

```
\documentclass{article}

\begin{document}

\section{My life}

I've read only two books in my life \cite{Bekke92}, \cite{Brodie84}.

\begin{thebibliography}{Brodie84}

\bibitem[Bekke92]{Bekke92}
  J.H.~ter~Bekke, \emph{Semantic datamodeling}, Prentice Hall, Hemel
  Hempstead, ISBN~0-13-806050-9, 1992.

\bibitem[Brodie84]{Brodie84}
  L.~Brodie, \emph{Thinking Forth, a language and philosophy for
  solving problems}, Prentice Hall, ISBN 0-13-917568-7, 1984.

\end{thebibliography}

\end{document}
```

Entries are started with `\bibitem` and appear in the `bibliography` environment environment. Entries are referred to with the `\cite` command.

In ConTeXt, bibliographies are supported by the `bib` module. This module is not yet in the standard release, you have to download it separately at `http://www.cybercomm.nl/~bittext/temp/m-bib.zip`. See section 11.2 on how to generate the documentation for this module. It seems the documentation is a bit ahead of the implementation, but expect the module to improve over the course of the next few months.

In ConTEXt, references have to be in a separate file. For this example it is easiest to have them in a file with the same name, but with the `.bbl` extension.

An example of such a file is:

```
\startpublication[k=Brodie84,
                  t=article,
                  a=L.~Brodie,
                  y=1984,
                  s=LB84]
\artauthor[]{Leo}[L.]{}{Brodie}
\arttitle{Thinking Forth, a language and philosophy for solving
          problems}
\journal{Prentice Hall}
\pubyear{1984}
\stoppublication

\startpublication[k=Bekke92,
                  t=article,
                  a=J.H.~ter~Bekke,
                  y=1992,
                  s=JB92]
\artauthor[]{Johan}[J.H.]{ter}{Bekke}
\arttitle{Semantic datamodeling}
\journal{Prentice Hall}
\pubyear{1992}
\stoppublication
```

Entries are started with the `\startpublication` command.

Having this file, it is now possible to use them as follows:

```
\usemodule[bib]

\setuppublications
  [numbering=yes,
  sort=author]

\starttext

\section{My life}

I've read only two books in my life \cite[Bekke92,Brodie84]. But I've
admit that I no longer think Forth \cite[Brodie84].

\completepublications

\stoptext
```

The command `\setuppublications` is optional. It defines such things as which entries to include, how to sort them or to include every entry or only the referenced ones. It currently does not seem possible to mimick the way LaTeX references things.

With `\completepublications` the list of publications is given. And of course there is also a `\placepublications` command which does not add something to the table of contents.

For bibTEX information, see section 8.4.

3.3 The description environment

LaTeX's `description` environment produces a list where each label is a keyword instead of a bullet or a number.

```
\documentclass{article}
\begin{document}

\begin{description}
\item[gnat] A small animal, found in the North Woods, that causes no
  end of trouble.
\item[gnu] A large animal, found in crossword puzzles, that causes no
  end of trouble.
\item[armadillo] A medium-sized animal.
\end{description}

\end{document}
```

In ConTeXt you use the `\definedescription` command to setup an environment. A description is not an enumeration like it is in LaTeX. In the following code we first define the environment, called `animal`. It's also possible to define the environment `description`, but that's quite confusing. In the main text we show how this new environment is used:

```
\definedescription
  [animal]
  [location=hanging,
   margin=standard,
   headstyle=bold]

\definestartstop
  [animals]
  [before=\blank\startpacked,
   after=\stoppacked\blank]

\starttext

\startanimals
\animal{gnat} A small animal, found in the North Woods, that
  causes no end of trouble.

\animal{gnu} A large animal, found in crossword puzzles, that
  causes no end of trouble.

\startanimal{armadillo}
A medium-sized animal.
\stopanimal

\stopanimals

\stoptext
```

Within `\startanimals`, descriptions (animals) can be given. You can use both `\animal` or `\startanimal` to define a description. The start/stop pair is of course more robust. The type `\animal` expects a `\par` (or empty line) to work.

The use of `\definestartstop` is optional, but this helps to clearly mark the definition of animals and to have a common point of settings like white space before and after. With `\blank` we get the default blank space before (the space between paragraphs). With `\startpacked` we get paragraphs that do not have white space between them.

A different solution, however without the hanging indent feature, is:

```
\starttext

\startitemize[4*broad,packed]
\sym{gnat} A small animal, found in the North Woods, that causes no
 end of trouble.
\sym{gnu} A large animal, found in crossword puzzles, that causes no
 end of trouble.
\sym{armadillo} A medium-sized animal.
\stopitemize
\stoptext
```

In this example, the text `armadillo` overlaps with its definition if the margin is not defined sufficiently large. So the first solution is better.

This example also shows ConTEXt's way of setting options. With `4*broad` we set the width of the symbol to four times the `broad` setting. With `packed` we specify that we don't want white space between paragraphs.

3.4 The enumerate environment

LaTeX's `enumerate` environment is used to produce a numbered list:

```
\documentclass{article}
\begin{document}

\begin{enumerate}
\item First item.
\item Second item.
\end{enumerate}

\end{document}
```

In ConTEXt lists, both numbered and unnumbered, are started with `\startitemize`. You use the `n` option to produce a numbered list:

```
\starttext

\startitemize[n]
\item First item.
\item Second item.
\stopitemize

\stoptext
```

See also the example in section 3.5. See section 7.9 for how to influence the layout of lists.

3.5 The itemize environment

LaTeX's `itemize` environment is used to produce an unnumbered list:

```
\documentclass{article}
\begin{document}

\begin{itemize}
\item First item.
\item Second item.
\end{itemize}

\end{document}
```

Every line starts with a bullet.

In ConTeXt lists, both numbered and unnumbered, are started with `\startitemize`. If you don't use an option, you get an unnumbered list:

```
\starttext

\startitemize
\item First item.
\item Second item.
\stopitemize

\stoptext
```

See also the example in section 3.4. See section 7.9 for how to influence the layout of lists.

3.6 The quotation environment

You can quote someone in LaTeX with the `quote` environment:

```
\documentclass{article}
\begin{document}

\begin{quote}
  But letting go of the first is, to him, just a promise
  of pain with no promise of compensating pleasure.
\end{quote}

\end{document}
```

In ConTeXt you achieve the same effect with the `\startquotation` command:

```
\starttext

\startquotation
  But letting go of the first is, to him, just a promise
  of pain with no promise of compensating pleasure.
\stopquotation

\stoptext
```

LaTeX also has a `quotation` environment used for quotations of more than one paragraph. In ConTeXt you can always use `\startquotation`.

ConTeXt also has 'inline' quotes. Use either its `\quote` or `\quotation` command:

```
\starttext

This is a \quote{quote} and this is a \quotation{quotation}.

\stoptext
```

`\quote` surrounds your quote with single quote characters, `\quotation` surrounds your quote with double quote characters.

3.7 The verbatim environment

Verbatim text (text not subject to macro expansion) in LaTeX is done with the verbatim environment:

```
\documentclass{article}
\begin{document}

\begin{verbatim}
This is
```

```
verbatim \LaTeX.
\end{verbatim}

\end{document}
```

In ConTEXt you write this as:

```
\starttext

\starttyping
This is
verbatim \ConTeXt.
\stoptyping

\stoptext
```

4 Floats

Floats are pieces of text that do not follow the main flow, but can go on the same page or elsewhere. ConTEXt has very extensive support for them.

4.1 Figures

In LATEX you can include a bitmap if you use the `graphics` or `graphicx` package. With the `\includegraphics` command you can include a bitmap.

If you use pdftex, you can now define a figure as simply as:

```
\documentclass{article}

\usepackage[pdftex]{graphicx}

\begin{document}

\begin{figure}
\includegraphics{test.png}
\caption{Test picture}
\label{fig:test}
\end{figure}

\end{document}
```

This figure also has a caption and it has a label so you can refer to it. In ConTEXt you do this with:

```
\starttext

\placefigure
  []
  [fig:test]
  {Test picture}
  {\externalfigure[test.png]}

\stoptext
```

Probably the size of the figure differs between LATEX and ConTEXt. In ConTEXt figures have their natural size by default. You can influence the scaling of a figure with the scale option:

```
\starttext
```

```
\placefigure
  []
  [fig:test]
  {Test picture}
  {\externalfigure[test.png][scale=2000]}

\stoptext
```

The default scale is 1000 (100%). Scale 2000 gives you a figure twice as large. You can also specify the width of the figure in dimensions, for example half the text width (see example below).

In ConTeXt you usually define your figures at the top of your file, above the `\starttext` command. You can recall them when needed:

```
\useexternalfigure
  [testone]
  [test.png]
  [scale=2000]

\useexternalfigure
  [testtwo]
  [test.png]
  [width=.5\textwidth]

\starttext

\placefigure
  []
  [fig:testone]
  {Test picture: twice as large as it natural size}
  {\externalfigure[testone]}

\placefigure
  []
  [fig:testtwo]
  {Test picture: half as large as the text}
  {\externalfigure[testtwo]}

\stoptext
```

With `\useexternalfigure` the first command is the name of the macro you want to define, the next is the name of the file. The third parameter are scaling and sizing options.

In `http://www.pragma-ade.nl/zipped/bman-en.zip` you can find more things you can do with figures like placing two figures together or placing text left or right of a figure.

4.2 List of figures

A list of figures in LaTeX can be given with its `\listoffigures` command:

```
\documentclass{article}

\usepackage[pdftex]{graphicx}

\begin{document}

\listoffigures

\begin{figure}
\includegraphics{test.png}
```

```
\caption{Test picture}
\label{figure:test}
\end{figure}

\end{document}
```

In ConTEXt you use the `\completelistoffigures` command:

```
\starttext

\completelistoffigures

\placefigure
  [fig:test]
  {Test picture}
  {\externalfigure[test.png]}

\stoptext
```

Just like `\completecontent`, besides `\completelistoffigures` their also is a `\placelistoffigures` command which doesn't start a new page.

Be aware that `\completelistoffigures` only works at the beginning of a document, not at the end. There probably is a workaround, and as soon as I know it, I'll say so here.

5 Tables

5.1 The tabular environment

In LaTeX tables are defined with the `tabular` environment environment. A famous LaTeX example demonstrating many of its features is:

```
\documentclass{article}

\begin{document}

\begin{tabular}{|r||r@{--}l|p{1.25in}|}
\hline
\multicolumn{4}{|c|}{GG\&A Hoofed Stock}
  \\ \hline\hline
&\multicolumn{2}{c|}{Price}& \\ \cline{2-3}
\multicolumn{1}{|c||}{Year}
& \multicolumn{1}{r@{\,\vline\,}}{low}
& high & \multicolumn{1}{c|}{Comments}
  \\ \hline
1971 & 97 & 245 & Bad year for
           farmers in the west.   \\ \hline
  72 & 245 & 245 & Light trading due to a
                   heavy winter. \\ \hline
  73 & 245 & 2001 & No gnus was very
         good gnus this year.    \\ \hline
\end{tabular}

\end{document}
```

In ConTEXt tables are created within the `\starttable` environment:

```
\starttext

\starttable[|r|r|l|p(1.25in)|]
\HL
```

```
\VL \FOUR{GG\&A Hoofed Stock} \VL\SR
\HL
\VL \LOW{Year} \VL \TWO{Price}  \VL \LOW{Comments} \VL\SR
\DC            \DL[2]           \DC                \DR
\VL            \VL low \VL high \VL                \VL\SR
\HL
\VL 1971 \VL \TWO{97--245} \VL
  Bad year for farmers in the west. \VL\SR
\HL
\VL 72 \VL \TWO{245--245} \VL
  Light trading due to a heavy winter. \VL\SR
\HL
\VL 73 \VL \TWO{245--2001} \VL
  No gnus was very good gnus this year. \VL\SR
\HL
\stoptable

\stoptext
```

In ConTeXt you can use `\starttables` to create a table that can be split accross pages.. In LaTeX this is provided by the `longtable` environment, provided by an external package.

5.2 The tabbing environment

To align data vertically, one can use the `tabbing` environment in LaTeX. It's usage is quite complex, only a simple example is given:

```
\documentclass{article}

\begin{document}
\begin{tabbing}
  Armadillo: \= \kill
  Gnat: \> not edible \\
  Armadillo: \> not edible\\
\end{tabbing}
\end{document}
```

The same example in ConTeXt using its `\starttabulate` environment:

```
\starttext
\starttabulate[|l|p|]
\NC Gnat: \NC not edible \NC\NR
\NC Armadillo: \NC not edible \NC\NR
\stoptabulate
\stoptext
```

In LaTeX the first line more or less determines the format (but can be changed in latter lines). In ConTeXt the `\starttabulate` environment works more or less like its `\starttable` environment. The header specifies the format of the columns. The width of the columns will be equal to the row with the largest contents. You can also explicitly specify the width.

Every row starts with `\NC` (next column). Every row ends with a `\NC` and `\NR` (next row).

The difference between ConTeXt's `\starttabulate` and `\starttable` environments are that the former splits across pages. And the `p` column type is a bit smarter in the `\starttabulate` environment. The drawback of `\starttabulate` is, that it does not support vertical lines.

5.3 List of tables

In LaTeX one gets the list of tables with the `\listoftables` command. Only tables placed within the `table` environment environment are put in the table of contents.

In ConTeXt one uses the `\placelistofoftables` command. And as in LaTeX, only tables given as argument to `\placetable` are pu in the table of contents.

6 Math

6.1 General

Math in ConTeXt and LaTeX is not that different. Both depend mostly on TeX. LaTeX adds some environments for doing math. ConTeXt is mostly very close to TeX. In ConTeXt probably every TeX math command just works.

6.2 In-text formulas

In ConTeXt in-text formula's are produced by surrounding the formula by the $ character.

6.3 Display style formulas

In ConTeXt display style formula's are produced by surrounding the formula by two $ characters. You can also use the `\startformula` environment for exactly the same effect.

6.4 Numbered formulas

With LaTeX numbered formula's are made within the `equation` environment:

```
\documentclass{article}

\begin{document}
\begin{equation}
E = mc^2
\end{equation}
\end{document}
```

In ConTeXt numbered formulas are produced to prefix the display formula command with `\placeformula`.

```
\starttext
\placeformula $$ E = mc^2 $$
\stoptext
```

6.5 Multiline formulas

In LaTeX multiline formula's are produced with the `eqnarray` environment. A new line is started after the `\\` command.

In ConTeXt you use TeX's displaylines command. A new line is started after the `\cr` command.

```
\starttext

$$\displaylines{%
   ZeroOrOne = ( ( Amount\ mod\ Currency_RoundingFactor )\cr
  +\ Currency_Boundary )\cr
  div\ Currency_RoundingFactor}$$

\stoptext
```

6.6 Theorems and such

LaTeX has the `\newtheorem` command to define environments for theorems-like structures. You would use it as follows:

```
\documentclass{article}

\newtheorem{guess}{Conjecture}

\begin{document}
This is the first one:
\begin{guess}
  All conjectures are interesting.
\end{guess}
\end{document}
```

With ConTeXt you would use the `\defineenumeration` command. This command has many options, the following settings make it more or less equal to LaTeX's `\newtheorem`:

```
\defineenumeration
  [guess]
  [text=Conjecture,
  location=left,
  letter=it]

\starttext
This is the first one:
\guess All conjectures are interesting.

This is the second one:
\startguess Except this one. \stopguess
\stoptext
```

As can be seen both `\guess` and `\startguess` can be used. Also `\subguess`, `\subsubguess` and so on are now available.

ConTeXt's `\defineenumeration` can be used for any kind of thing you want to enumerate.

7 Changing the layout

As soon as you are able to write basic documents, you probably want to change their appearance. This chapter documents the differences between LaTeX and ConTeXt. As ConTeXt is a very, very flexible macro package, this is one of the largest chapters. Almost everything can be customized, you just call `\setup`*`something`*.

7.1 Page size

In LaTeX you give the page size as an option to the `\documentclass` command:

```
\documentclass[a4paper]{article}

\begin{document}
\end{document}
```

In ConTeXt you use the `\setuppapersize` command.

```
\setuppapersize[A4]

\starttext
\stoptext
```

With ConTeXt it is possible to do far more advanced things. You can typeset in A5 and print on A4 for example.

7.2 Fonts

Changing the font size in LaTeX is usually done with a document class option:

```
\documentclass[12pt]{report}

\begin{document}
\end{document}
```

In ConTeXt you use the `\setupbodyfont` command:

```
\setupbodyfont[12pt]

\starttext
\stoptext
```

Use `\setupbodyfont` only to set the document font. To switch to another font during typesetting, you should use `\switchtobodyfont`.

To switch to a postscript font in LaTeX, you can use certain packages. To switch to a Helvetica body font you would type:

```
\documentclass{article}
\usepackage{helvet}

\begin{document}
\end{document}
```

In ConTeXt you accomplish the same with the `\setupbodyfont` command:

```
\setupbodyfont[ber,phv,ss]

\starttext
\stoptext
```

With the option `ber` you specify that you want to use Karl Berry fontnames. With `phv` you specify that you want to load the Helvetica font definitions, and with `ss` you specify that you want to use a sans-serif font as the body font.

To switch to the default postscript fonts times, helvetica and courier, you would say:

```
\setupbodyfont[ber,pos]

\starttext
\stoptext
```

7.3 Interline spacing

To change the interline spacing in LaTeX you change the `baselineskip` variable:

```
\documentclass{article}

\setlength\baselineskip{12pt}

\begin{document}
\end{document}
```

This changes the distance between lines to 12pt, a good value for a 10pt font. As in LaTeX, if you change the size of the body font in ConTeXt, the line skip is automatically recalculated. You can set it yourself with `\setupinterlinespace`:

```
\setupinterlinespace[line=1.2\bodyfontsize]
```

```
\starttext
\stoptext
```

Be careful to always define the interlineskip in terms of the current `\bodyfontsize`, else you get unexpected results when the `\bodyfontsize` changes, for example in chapter headings.

7.4 Spacing between paragraphs

To change the interparagraph spacing in LaTeX, you change the `\parskip` variable:

```
\documentclass{article}

\setlength{\parskip}{3pt}

\begin{document}
This is my first paragraph.

This is my second paragraph.
\end{document}
```

This gives you 3 extra points of white space between paragraphs. In ConTeXt you use the `\setupwhitespace` command.

```
\setupwhitespace[3pt]

\starttext
This is my first paragraph.

This is my second paragraph.
\stoptext
```

However, instead of a fixed size specification, it is much better to use current font size related specifications like `medium` or `big`:

```
\setupwhitespace[medium]

\starttext
This is my first paragraph.

This is my second paragraph.
\stoptext
```

7.5 Paragraph indentation

In LaTeX every paragraph has some white space at the beginning of its first line. You can disable this by setting the amount of white space to 0:

```
\documentclass{article}

\setlength{\parindent}{0pt}

\begin{document}
This is my first paragraph. This is my first paragraph. This is my
first paragraph. This is my first paragraph. This is my first
paragraph. This is my first paragraph.

This is my second paragraph.
\end{document}
```

ConTEXt doesn't start with white space. To start a paragraph with some white space say:

```
\setupindenting[medium]

\starttext
This is my first paragraph. This is my first paragraph. This is my
first paragraph. This is my first paragraph. This is my first
paragraph. This is my first paragraph.

This is my second paragraph.
\stoptext
```

7.6 Position of the page number

In LaTeX the position of the page number can be set with the `\pagestyle` command. In ConTEXt you use the `\setuppagenumbering` command.

There are four LaTeX page styles. The corresponding ConTEXt settings are given below:

1. `\pagestyle{plain}`:

```
\setuppagenumbering[location={footer,middle}]
```

2. `\pagestyle{empty}`:

```
\setuppagenumbering[location=]
```

3. `\pagestyle{headings}`:

```
\setuppagenumbering[location={header,middle}]
```

4. `\pagestyle{myheadings}`:

```
\setuppagenumbering[location=header]
```

(see ....)

Note that in ConTEXt a page consists of much more than LaTeX's three default units *head*, *body* and *foot*, so you have much more options.

7.7 Roman and arabic pagenumbers

In LaTeX Roman and Arabic page numbers are specified with the `\pagenumbering` command.

```
\documentclass{article}

\begin{document}

\pagenumbering{roman}
\tableofcontents

\chapter{Introduction}
\pagenumbering{arabic}
This is a test.

\end{document}
```

In ConTEXt you use the `\setupnumbering` command.

```
\startfrontmatter
```

```
\setuppagenumbering[conversion=romannumerals]
\setuppagenumber[number=1]
\completecontent

\stopfrontmatter

\startbodymatter

\setuppagenumbering[conversion=numbers]
\setuppagenumber[number=1]
\chapter{Introduction}
This is a test.

\stopbodymatter
```

This example also shows a typical document structure by its use of `\startfontmatter` and `\startbodymatter`. The front matter has roman page numbers, the body matter arabic ones. We also use `\setuppagenumber` to start counting from 1 for each of the sections. The first `\setuppagenumber` could be omitted, but typically you have a front page and such, so page numbering starts a bit later than the first physical pages.

7.8 Setting up chapter and sections

In LaTeX changing the format of chapters and sections is not possible unless you change LaTeX internal commands. Therefore, no examples of doing this in LaTeX are given, but only the ConTeXt options are demonstrated.

Every section heading can be setup with ConTeXt's `\setuphead` command. Probably one of the first things you want to change is the font used for the chapters and sections.

```
\setuphead[chapter][letter={\switchtobodyfont[20pt,ss]\bf}]
\setuphead[section][letter={\switchtobodyfont[16pt,ss]\bf}]

\starttext

\chapter{This is a chapter}

First sentence.

\section{This is a section}

First sentence.

\stoptext
```

This gives you a 20 points, bold, sans serif font for chapter headings.

It is possible to define that you want to start a chapter on a right page:

```
\setuppagenumbering[alternative=doublesided]
\setuphead[chapter]
  [page=right]

\starttext

\chapter{First thought}

First sentence.

\chapter{Second thought}

Second sentence.

\stoptext
```

You also need to turn doublesided pagenumbering on with `\setuppagenumbering`. Now every chapter starts on an uneven numbered page.

The following example changes even more. The number is set in a different font, and before and after the chapter is set, we execute our own commands.

```
\setuphead
  [chapter]
  [numberstyle=bold,
  textstyle=cap,
  before=\hairline\blank,
  after={\nowhitespace\hairline\blank[line]}]

\starttext

\chapter{This is a chapter}

A sentence.

\stoptext
```

Note that almost any(!) setup command has before and after options, so really everything in ConTEXt can be changed easily.

It is also possible to format the entire section heading yourself. The following example formats the subsection heading. You need to write a macro which expects two parameters: the number of that section and the title of that section. And then you're on your own.

```
\setuphead
  [subsection]
  [command=\myhead]
\def\myhead#1#2{#2}

\starttext

\chapter{This is a chapter}

\section{Section}

\subsection{Yes}

A sentence.

\stoptext
```

Note that turning off the number is a standard option, so this setup can also be accomplished with:

```
\setuphead
  [subsection]
  [number=no]

\starttext

\chapter{This is a chapter}

\section{Section}

\subsection{Yes}

A sentence.

\stoptext
```

7.9 Setting up lists

Enumerations in ConTeXt have white space between each items. You can disable this by using the packed option.

```
\setupitemize[packed]

\starttext
\startitemize[n]
\item First line.
\item Second line, no white space above it.
\stopitemize
\stoptext
```

8 References

8.1 In document references

8.2 Cross document references

8.3 Index

In LaTeX producing indexes is a two step process. In the preamble you put the `\makeindex` command so an index file (`.idx`) is created. The contents of every `\index` command is put into that file. Next the `makeindex` command is used to produce the actual index which can be included in your document with the `\input` command.

```
\documentclass{article}

\makeindex

\begin{document}

Put \index{this} and \index{that} in the index.

\input{testlatex.ind}

\end{document}
```

In ConTeXt creating indexes is also a two step process, but this is transparant if you use `texexec` to compile your documents. With the `\index` command you put entries in the index, with `\completeindex` you get the entire index at that point.

```
\starttext

Put \index{this} and \index{that} in the index.

\completeindex
\stoptext
```

8.4 bibtex

You can use `bibtex` to produce `.bbl` files as usual. The new `bib` module (see section 3.2) can read `.bbl` files fine. You can specify on or more databases in `\setuppublications` after the database keyword.

```
\usemodule[bib]

\setuppublications
  [database={mybibs},
  numbering=yes,
  sort=author]
```

```
\starttext

\section{My life}

I've read only two books in my life \cite[laan95:_types_pascal].
But I've admit that I no longer think Forth \cite[wogan:73a].

\completepublications

\stoptext
```

However, bib support is not yet foolproof.

9 Interactive documents

9.1 Defining an interactive document

```
\usepackage{article}
\usepackage
  [pdftitle={Test document},
   pdfauthor={Berend de Boer},
   pdfsubject={There is more than LaTeX},
   pdfkeywords={LaTeX ConTeXt TeX},
   colorlinks,
   linkcolor=blue]
  {hyperref}

\begin{document}
\end{document}
```

```
\setupinteraction
  [status=start,
   title={Test document},
   author={Berend de Boer},
   subtitle={There is more than LaTeX},
   keywords={LaTeX ConTeXt TeX},
   color=blue]

\starttext
\stoptext
```

9.2 References which are URLs

In LaTeX the hyperref package provides hyperlinking support. To generate a URL reference, you use the `\href` command:

```
\documentclass{article}

\usepackage{hyperref}

\begin{document}

The \href{http://www.freebsd.org}{greatest server OS} on this world.

\end{document}
```

```
\setupinteraction[state=start]
```

```
\useURL
  [contextsupport]
  [http://www.pragma-ade.nl]

\starttext

The \ConTeXt\ support site is \from[contextsupport].

\stoptext
```

10 Other

10.1 Title page

10.2 Text in margin

10.3 Color

You can enable color with the `\setupcolors` command.

```
\setupcolors[state=start]

\starttext
\stoptext
```

10.4 Babel

10.5 Interfaces

10.6 Breaking lines

10.7 Vertical white space

If you need vertical white space, LaTeX has the `\vspace` command. This command only works between paragraphs. At the beginning and end of a page white space ‘disappears’, use `\vspace*` for white space that does not disappear.

```
\documentclass{article}

\begin{document}
\vspace*{5cm}
This line comes first.

\vspace{2\baselineskip}

This line comes second.

\vspace{.5\baselineskip}

And this is the third line.

\end{document}
```

In ConTeXt the `\blank` command gives blank space. It also ends the preceding paragraph. Use the `force` option to force white space at the beginning and end of a page.

```
\starttext
\blank[5cm,force]
This line comes first.
\blank[2*line]
This line comes second.
\blank[medium]
```

```
And this is the third line.
\stoptext
```

11 Understanding ConTEXt

11.1 texexec

ConTEXt uses the external program `texexec` to do lots of things which are more difficult to do directly in TEX:

1. ConTEXt writes its table of contents entries to a `.tui` file. If the compilation went fine, `texexec` copies this to the corresponding `.tuo` file.
   This means that when a compilation does not run to completion (you cancel it for example), the table of contents entries and references have not disappeared.
2. `texexec` sorts indexes and sorted lists.

11.2 Module documentation

Generate ConTEXt module documentation with:

```
texexec --module m-bib
```

You need to have the documentation styles which can be downloaded from `http://www.pragma-ade.nl`, filename `cont-doc.zip`.

12 LaTeX commands

List of LaTeX commands used in this document.

13 ConTEXt commands

List of ConTEXt commands used in this document.

14 LaTeX to ConTeXt reference

14.1 Standard environments

A list of all LaTeX environments and the corresponding ConTeXt environment is provided below. Note that LaTeX environments are written as:

```
\begin{environment}
\end{environment}
```

In ConTeXt you write this as:

```
\startenvironment
\stopenvironment
```

LaTeX environment	ConTeXt environment	Remark
abstract	no equivalent.	
array	no equivalent.	
center	alignment	
description		See section 3.3
displaymath		
document	text	
enumerate	itemize	
equation	placeformula	
eqnarray	displaylines	Use the standard TeX commands.
figure		Use placefigure.
flushleft	alignment	
flushright	alignment	
itemize	itemize	
letter	no equivalent.	
list	itemize	
math		Use $ before and $ after the formula.
minipage	no equivalent.	
note	no equivalent.	
overlay	no equivalent.	
picture	MPgraphic	In ConTeXt you can include metapost code to achieve the same and better effects.
quote	citaat	
quotation	citaat	
slide	no equivalent.	
sloppypar		
tabbing	tabulate	
table	table	Use together with placetable
tabular	table	
titlepage	no equivalent.	
thebibliography	publication	Requires the `bib` module.
theindex		Use completeindex.
trivlist	itemize	
verbatim	typing	
verse		

14.2 Standard commands

Many LaTeX commands come from third party packages. Most of the things they provide (or fix) are standard features in ConTeXt.

LaTeX command	ConTeXt command	Remark
setlanguage		Provided by the babel package.

14.3 Environments provided packages

Many LaTeX environments come from third party packages. Most of the things they provide (or fix) are standard features in ConTeXt.

LaTeX package	ConTeXt environment	Remark
comment	comment	Comment out certain parts of your document so it does not show up in the .dvi or .pdf file. LaTeX's `comment` environment is part of Rainer Schöpf's verbatim package.
longtable	tables	The longtable package provides tables that can be broken accross page.