MAPS

NUMMER 33 • NAJAAR 2005

NTG NEDERLANDSTALIGE TEX GEBRUIKERSGROEP

NTG NEDERLANDSTALIGE TEX GEBRUIKERSGROEP

Voorzitter
Hans Hagen
pragma@wxs.nl

Secretaris
Willi Egger
w.egger@boede.nl

Penningmeester
Wybo Dekker
wybo@servalys.nl

Bestuursleden
Maarten Wisse
Maarten.Wisse@urz.uni-hd.de

Frans Goddijn
frans@goddijn.com

Karel Wesseling
k.h.wesseling@planet.nl

Postadres
Nederlandstalige TEX
Gebruikersgroep
Maasstraat 2
5836 BB Sambeek

Postgiro
1306238
t.n.v. NTG, Deil
BIC-code: PSTBNL21
IBAN-code: NL05PSTB0001306238

E-mail bestuur
ntg@ntg.nl

E-mail MAPS redactie
maps@ntg.nl

WWW
www.ntg.nl

De **Nederlandstalige TEX Gebruikersgroep (NTG)** is een vereniging die tot doel heeft de kennis en het gebruik van TEX te bevorderen. De NTG fungeert als een forum voor nieuwe ontwikkelingen met betrekking tot computergebaseerde documentopmaak in het algemeen en de ontwikkeling van 'TEX and friends' in het bijzonder. De doelstellingen probeert de NTG te realiseren door onder meer het uitwisselen van informatie, het organiseren van conferenties en symposia met betrekking tot TEX en daarmee verwante programmatuur.
De NTG biedt haar leden ondermeer:

□ Tweemaal per jaar een NTG-bijeenkomst.
□ Het NTG-tijdschrift MAPS.
□ De 'TEX Live'-distributie op DVD/CDROM inclusief de complete CTAN softwarearchieven.
□ Verschillende discussielijsten (mailing lists) over TEX-gerelateerde onderwerpen, zowel voor beginners als gevorderden, algemeen en specialistisch.
□ De FTP server `ftp.ntg.nl` waarop vele honderden megabytes aan algemeen te gebruiken 'TEX-producten' staan.
□ De WWW server `www.ntg.nl` waarop algemene informatie staat over de NTG, bijeenkomsten, publicaties en links naar andere TEX sites.
□ Korting op (buitenlandse) TEX-conferenties en -cursussen en op het lidmaatschap van andere TEX-gebruikersgroepen.

Lid worden kan door overmaking van de verschuldigde contributie naar de NTG-giro (zie links); vermeld IBAN- zowel als SWIFT/BIC-code en selecteer shared cost. Daarnaast dient via `www.ntg.nl` een informatieformulier te worden ingevuld. Zonodig kan ook een papieren formulier bij het secretariaat worden opgevraagd.
De contributie bedraagt € 40; voor studenten geldt een tarief van € 20. Dit geeft alle lidmaatschapsvoordelen maar *geen stemrecht*. Een bewijs van inschrijving is vereist. Een gecombineerd NTG/TUG-lidmaatschap levert een korting van 10% op beide contributies op. De prijs in euro's wordt bepaald door de dollarkoers aan het begin van het jaar. De ongekorte TUG-contributie is momenteel $65.

MAPS bijdragen kunt u opsturen naar `maps@ntg.nl`, bij voorkeur in LaTEX- of ConTEXt formaat. Bijdragen op alle niveaus van expertise zijn welkom.

Productie. De Maps wordt gezet met behulp van een LaTEX class file en een ConTEXt module. Het pdf bestand voor de drukker wordt aangemaakt met behulp van pdfetex 1.30 Web2C 7.5.5 draaiend onder Linux 2.6. De gebruikte fonts zijn Bitstream Charter, schreefloze en niet-proportionele fonts uit de Latin Modern collectie, en de Euler wiskunde fonts, alle vrij beschikbaar.

TEX is een door professor Donald E. Knuth ontwikkelde 'opmaaktaal' voor het letterzetten van documenten, een documentopmaaksysteem. Met TEX is het mogelijk om kwalitatief hoogstaand drukwerk te vervaardigen. Het is eveneens zeer geschikt voor formules in mathematische teksten.
Er is een aantal op TEX gebaseerde producten, waarmee ook de logische structuur van een document beschreven kan worden, met behoud van de letterzetmogelijkheden van TEX. Voorbeelden zijn LaTEX van Leslie Lamport, AMS-TEX van Michael Spivak, en ConTEXt van Hans Hagen.

Inhoudsopgave

Redactioneel

Voor u ligt het 33^e nummer van de Maps, met 78 pagina's niet zo dik deze keer, maar de kwaliteit is er niet minder om. Ook nu weer is de tweede helft ervan in kleur uitgevoerd, en ook deze keer is er weer een grote variatie: er zijn artikelen over CTAN, fonts, cd-hoesjes, woordafbreking, kleurscheiding, PowerPoint (pardon: PowerDot)-presentaties, stroomdiagrammen en installatie.

□ Maarten Sneep *(Wachten op een Ca-tas-tro-fe)* In het kader van de zoveelste spellingwijziging geeft Maarten Sneep zijn ongezouten mening. Veel daarvan zal u bekend voorkomen. Maar veel was ook nog niet bij opgekomen. Geniet van de bevestiging van uw eigen verontwaardiging.

□ Dennis van Dok *(Jewel case listings for mp3 cdroms)* laat zien hoe zo'n 130 nummers op een mp3 muziek-cd op het formaat van een *jewel case* toch nog overzichtelijk kunnen worden weergegeven. Maar ook hij heeft geen antwoord op de vraag hoe dat nu moet met de, pakweg, 1000 nummers die op een dvd passen. Gelukkig is de tijd niet ver meer dat de achterkant van zo'n juwelendoos gewoon een 1600x1200 flatscreen is met een in de rug ervan ingebouwde processor waarmee door de vele pagina's gebladerd kan worden.

□ Siep Kroonenberg *(Font installation the shallow way)*: Siep pakt een van de lastigste TEX-problemen aan: hoe een nieuw font te installeren zonder door bergen foutmeldingen te worden overspoeld. Haar pragmatische oplossing blijkt verrassend eenvoudig. Maak er eens een opwindende menukaart mee en nodig uw TEX-collega's aan de dis.

□ Idris Samawi Hamid *(Installing Expert Fonts: Minion Pro)* pakt de font-installatie minder pragmatisch aan, maar gaat dan ook in op alle details van het minion font.

□ Hans Hagen *(Hyphenation Patterns)* vertelt ons de ins en outs van hyphenation in ConTEXt. Als u geen context-gebruiker bent, sla het dan niet meteen over – het lijkt erop dat dit ook betekenis kan hebben voor andere formats.

□ Taco Hoekwater *(What do you do with ConTEXt?)* heeft een enquête gehouden onder ConTEXt-gebruikers met de vraag: wat doe er er eigenlijk mee. Een selectie uit de antwoorden.

□ Piet van Oostrum *(Een uittreksel uit de recente bijdragen in het CTAN archief)* maakt ons weer attent op nieuwe CTAN bijdragen; deze keer met veel aandacht grafische applicaties, maar ook voor floats, scheikundeformules, sudoku, muziek en nieuwe Europese lidstaten. En tenslotte probeert hij u te belemmeren in het bestrijden van uw rookverslaving.

□ Jan van de Craats *(Color separation in two-color printing)* verhaalt over de drukkersproblemen die als gevolg van kleurscheiding kunnen ontstaan, zoals het verschijnen van witte schaduwen aan de randen van schrifttekens. Hij heeft er gelukkig ook een oplossing voor.

□ Hendri Adriaens en Chris Ellison *(Powerdot)* biedt een volwaardig alternatief voor PowerPoint. Dit verhaal is nog slechts een beschrijving van hoe zij het pakket in de zomervakantie ontwikkelden – een zomerzotheid zeg maar. U zult er in een volgende maps ongetwijfeld meer over horen.

□ Siep Kroonenberg *(Managing a network TEX installation under Windows)* laat zien hoe zij op de economische faculteit van de RUG gebruikers centraal voorziet van een MikTeX installatie, met TeXnicCenter als editor en frontend.

□ Ovidiu Gheorghieş *(An Introduction to MetaUML)*: wie zich bezighoudt met het maken van stroomdiagrammen kan hier zijn hart ophalen. MetaPost en UML (Unified Modeling Language) komen hier samen om zulke diagrammen zeer gestructureerd samen te stellen.

De redactie wenst u weer veel inspiratie en veel leesplezier.

Wybo Dekker
wybo@servalys.nl

Wachten op een Ca-tas-tro-fe

Het kan u na zaterdag 17 december 2005 niet ontgaan zijn: er komt *weer* een nieuwe spelling aan. Het stof van de wijzigingen uit 1995 was nog nauwelijks neergedaald, tenminste in mijn hoofd, maar er worden nu nog enkele plooien gladgestreken. We mogen deze veranderingen geen “spellingwijziging” noemen, want deze aanpassing is niet zo ingrijpend – tenminste dat vindt de Nederlandse-taalunie. Als ik een beperkte lijst met veranderingen doorneem, rijzen mij echter de haren te berge. Sommige wijzigingen geven aardig de huidige maatschappelijke veranderingen weer: “kerstman” moet nu met een hoofdletter terwijl “Christus” gedecapiteerd is. Bij andere wijzigingen begint de betekenis van een woord te veranderen: Tweede-Kamervoorzitter verliest het streepje, en degradeert daarmee ofwel tot een vice-voorzitter van zichzelf, of het wordt de opvolger van de eerste voorzitter. De partij van de huidige voorzitter neemt straks mogelijk plaats in een coalitie met een stel “vruchten gebakjes”: “appèl” verliest het accent en wordt “appel” – eet smakelijk.

Als ambtenaar dien ik me aan die nieuwe spelling te houden, al ben ik blij dat het grootste deel van mijn geschreven communicatie in het Engels plaats vindt – het voordeel van werken in het internationaal wetenschappelijk onderzoek. U mag uw gang gaan, en ik kan u van harte aanbevelen om te schrijven zoals u dat geleerd heeft, en uw creativiteit niet te laten dwarsbomen door een stel *taalmalloten*.

Er zijn echter een paar zaken waar we niet aan kunnen ontkomen: Het onderwijs moet de nieuwe spelling volgen, gedwongen door de centrale examens en de inspectie. Ik heb medelijden met iedereen die nu ongeveer twintig is: dit wordt de *derde* spelling die zij moeten leren, geen wonder dat de spelvaardigheid achteruit holt. Bovendien zal het enige tijd vergen voordat alle lesmethoden vervangen zijn, waardoor leerlingen vanaf het eerste begin geconfronteerd worden met een “foutief” woordbeeld. Geld om de lesmethoden versneld te vervangen is er uiteraard niet. Mijn medelijden moet zich ook uitstrekken tot de leerkrachten die het allemaal moeten overdragen: weer een nieuwe set regeltjes met een poging om bij elke verzonnen wijziging een volslagen willekeurige verklaring te verzinnen. Taal is *gegroeid*, en valt niet logisch te verklaren, elke poging daartoe kan uitsluitend leiden tot verwarring. Bovendien is taal een levend iets, en dienen de regels de praktijk te volgen. In dit geval volgt de praktijk de regels. Dat was al duidelijk met de tussen-*n*, die de uitspraak na 1995 duidelijk gewijzigd heeft.

Wat heeft dit met TEX te maken? Alles en niets. U kunt nog steeds schrijven wat u wilt, u kunt de spelling van enkele grote dagbladen en tijdschriften volgen – zij boycotten immers ook de nieuwe spelregels. Aan de andere kant zult u er last van ondervinden. Eén van de prettige eigenschappen van TEX is juist dat de woordafbrekingen automatisch goed gaan, mits u de juiste taal instelt. En daar zit het probleem: naast het toevoegen of weglaten van koppeltekens, veranderen ook de afbreekregels voor een flink aantal woorden. De ca-tas-tro-fe in de titel is geen grap, dat is volgens de nieuwe regels *correct*. Ik weet niet hoe het met uw woordbeeld zit, maar ’t mijne is aan gruzelementen met dergelijke wijzigingen. Woordafbrekingen aan het einde van een regel kunnen al struikelblokken zijn, maar als het ook nog eens op een plaats gebeurt die de hele historie van een woord overboord gooit, dan leidt dat zeker tot valpartijen.

Op de laatste NTG bijeenkomst hebben we van Piet van Oostrum kunnen horen wat de status van de woordenlijst is die de basis vormt voor onze afbreekpatronen. *We*, en daarmee bedoel ik alle mensen die Nederlands spreken, kunnen niet vrij beschikken over de woordenlijst in het “Groene boekje”, het copyright daarvan ligt bij de Sdu. Dit is haast exemplarisch voor de situatie van het Nederlands: het is ons ontnomen. Uitgeverijen van schoolboeken willen echter graag gebruik maken van TEX en andere open source software, zoals OpenOffice. Dat stuit echter op een probleem: Op dit moment is er geen gecertificeerde woordenlijst voor gebruik met open source software, en uitgeverijen zijn verplicht om te controleren dat hun schoolboeken aan de officiële spelling voldoen – hoe absurd die spelling ook is. Het plan is dan ook om een dergelijke woordenlijst te produceren, en te laten certificeren door de Nederlandse-taalunie. Gezin de absurditeit van de wijzigingen neem ik aan dat het aantal vrijwilligers om deze regels daadwerkelijk door te voeren beperkt is. Ik zorg in elk geval voor een extra kopie van de huidige lijst, die is immers beschikbaar onder de LGPL[1]. Dit neemt overigens niet weg dat het opnieuw bekijken en corrigeren van de woordenlijst een nuttige bezigheid kan zijn, als is het maar om de lijst in de huidige spelling aan te vullen en te verbeteren. Ook het uitsplitsen naar stamwoorden, en op termijn misschien zelfs toevoegen van woord-soort kenmerken vergroot de waarde van de woordenlijst voor gebruik in allerlei computer-software. Deze activiteiten zijn echter onafhankelijk van een wijziging in de spelling. Het terugdraaien van een verandering in de spelling is overigens al eerder vertoond, ook in Duitsland is een spellingwijziging gedeeltelijk teruggedraaid wegens aanhoudend protest. Dat protest zal voorlopig niet verstommen: de sectie Nederlands van de vereniging van leraren van de levende talen (VLLT) sluit niet uit dat ze de wijzigingen zullen boycotten[2].

Ik weet zeker dat een depressie zich van mij meester zal maken als weer een hogedrukgebied vanuit Frankrijk onze kant op komt. Niet dat foute afbrekingen voorheen niet voorkwamen, maar dan wist je altijd nog: "Dit is waarschijnlijk een fout in de software, of een foutieve instelling". De gedachte dat iemand over deze wijziging heeft nagedacht en dit vervolgens heeft goedgekeurd is haast onverdraaglijk. Toch 'De Slegte' maar eens bezoeken voor een oude "Van Dale". Heeft iemand nog een afbraak-bestandje van vóór 1995?

Noten

[1] De GNU "Lesser General Public license", kopiëren en gebruiken mag, maar wijzigingen moeten altijd weer beschikbaar gesteld worden onder dezelfde voorwaarden.
[2] NRC Handelsblad, 20 december 2005

Maarten Sneep
`maarten.sneep@xs4all.nl`

De Volkskrant:
http://nieuwsbrief.taalpost.nl/r/tp.plx?1050-480
Trouw:
http://nieuwsbrief.taalpost.nl/r/tp.plx?1051-480
NRC Handelsblad:
http://nieuwsbrief.taalpost.nl/r/tp.plx?1052-480
Elsevier:
http://nieuwsbrief.taalpost.nl/r/tp.plx?1053-480
De Telegraaf:
http://nieuwsbrief.taalpost.nl/r/tp.plx?1054-480
Vlaamse kwaliteitskranten:
http://nieuwsbrief.taalpost.nl/r/tp.plx?1055-480
De Standaard:
http://nieuwsbrief.taalpost.nl/r/tp.plx?1056-480
Taalunie:
http://nieuwsbrief.taalpost.nl/r/tp.plx?1057-480
Werkgroep Spelling:
http://nieuwsbrief.taalpost.nl/r/tp.plx?1058-480
Minister van Onderwijs:
http://nieuwsbrief.taalpost.nl/r/tp.plx?1059-480
Hans Heestermans, ex-hoofdredacteur Van Dale:
http://nieuwsbrief.taalpost.nl/r/tp.plx?1060-480
Rob Schouten in middeleeuwse spelling:
http://nieuwsbrief.taalpost.nl/r/tp.plx?1061-480
Genootschap Onze Taal:
http://nieuwsbrief.taalpost.nl/r/tp.plx?1062-480
Henk Verkuyl, Emeritus
http://www.let.uu.nl/˜henk.verkuyl/personal/spelling.html

Jewel case listings for mp3 cdroms

Abstract
Making jewel case listings for mp3 cdroms is a particular challenge, since up to about ten times as much information has to be on them as on jewel cases for regular audio disks. Here TEX's abilities to adjust entire paragraphs, as opposed to just single lines, shine at you.

Keywords
mp3, jewel case, paragraph filling, grip, ripping, audio

Introduction

The use of MP3 audio compression brings a dramatic increase in music storage capacity of digital media. Where a music Compact Disc is limited to about 70 minutes of music, a data CD will store around ten times that much in high quality MP3s (128 kbit, 44.1 kHz stereo encoding).

With more and more devices hitting the market that can read MP3 CDs, it becomes attractive to take one's CD collection, rip and encode it to MP3 and burning it on several data CDs. The applications are numerous, even if I only mention playing MP3s during parties. You need fewer CD changes, resulting is less musical downtime.

The creation of MP3 CDs brings some challenges, and this article will face two of them. They are:

1. the extraction of track information from the MP3 files, to create a listing of the entire contents of the CD, and
2. the formatting of this listing so that it fits the backside of a standard CD case ("jewel case") and still looks nice.

The formatting is a typographical problem, so it's no surprise that TEX comes into play there. But TEX also shows its strength as a word processor when it parses the generated contents list.

The listing will include track number, name, and duration; this information can easily be extracted from the MP3 files themselves if the CD ripping software put it in. A small ruby program is displayed that generates the TEX input from a directory of MP3 files. The format of the TEX input file is simple enough to generate with your programming language of choice.

The typesetting is done with LaTEX, but plain TEX users will appreciate the fact that the solution uses mainly basic constructs. Some changes will have to be made to font selection and multicolumn formatting (done with `multicols`) if another format than LaTEX is to be used.

```
#!/usr/bin/env ruby

# mp3totex - list mp3's in given directory in TeX compatible format

require 'rubygems'
require 'mp3info'

album, tracks = nil, []

Dir["#{ARGV[0]}/*.mp3"].sort.each do |f|
  m = Mp3Info.new(f)
  album = m.tag['album'] unless album
  ttit = m.tag['title']
  tlen = m.length
  tnum = m.tag1['tracknum']
  tracks[tnum-1] = "%03d %s...(%d:%02d).\n" %
                   [tnum, ttit, tlen/60, tlen%60 ]
end
puts "\\title #{album}.",tracks
```

Figure 1. The ruby program that generates the input file.

The Playlist

There is a plethora of software available to create MP3s of music CDs. I use Grip (`http://nostatic.org/grip/`), which automates the entire process. Upon inserting a music CD, Grip checks with an on-line CDDB database for disc and track information. This information will be stored in the ID3 tags in the MP3s themselves. Grip also generates playlists in the form of `m3u` files, which are text files listing the path names of the generated MP3s.

Although most MP3 playing *software* can handle m3u playlists, *hardware* players are oblivious to this concept. They don't handle directory structures very well and tend to play tracks in 'natural' order (as layed out on the disc) or alphabetic order. Therefore, the best option is to prefix the MP3 file names with the three digit number indicating the order in which they are to appear. This number will also show up in the final track listing.

So, after ripping and burning I have:

1. a data CD containing ten music CDs worth of MP3s,
2. ten corresponding `m3u` files listing the names of the files.

For reasons explained later, I want the TEX input to come out looking like this:

```
\title I Robot.
012 I Robot...(6:03).
013 I Wouldn't Want to Be Like You...(3:23).
014 Some Other Time...(4:05).
015 Breakdown...(3:52).
016 Don't Let It Show...(4:21).
017 The Voice...(5:23).
018 Nucleus...(3:31).
019 Day After Day (The Show Must Go on)...(3:49).
020 Total Eclipse...(3:09).
021 Genesis Ch.1. V.32...(3:28).
```

All lines are delimited by a period. The tracks have a number, then a space, the title, three dots and the track time in parentheses. This formatting must be followed quite

```
052  A Forest The Cure . . . (4:55)
053  Fortune Presents Gifts
     not According to the
     Book Dead can Dance  (6:03)          (1)
054  Signs of Life Pink Floyd
     . . . . . . . . . . . . . . . . . (4:24)
```

```
052  A Forest The Cure . . . . . . . . . . (4:55)
053  Fortune Presents Gifts not Ac-
     cording to the Book Dead can          (2)
     Dance . . . . . . . . . . . . . . . . . . . (6:03)
054  Signs of Life Pink Floyd  . . . . . (4:24)
```

Figure 2. Example listing at two different widths.

strictly. No white space should be placed between the dots and the parentheses, for instance.

The Ruby program in Figure 1 produces this input from the directory of MP3 files. Here is a short description of what it does. Line 6 imports the mp3info module that is capable of reading the tags of an MP3 file. On line 10 a loop starts over all the MP3 files in the directory that was given as a command-line argument. The tags are extracted and stored formatted in the `tracks` array, in the proper order. The album is recorded only once, which means that all files are considered to be from the same album. After the loop, the album title and the tracks are printed.

If you run the program on each album directory and append the outputs to a single file, you should now have all your tracks nicely listed in the proper format. Now is the time to do some last-minute manual touch-ups that an automatic procedure can not address.

The Layout

The main challenge lies in fitting over a hundred track numbers, titles and times in the cramped space of a CD case, which measures 13.7cm × 11.7cm, and still make browsing the list a pleasant experience. This cannot be done without using a very economic typeface; I chose Helvetica narrow for the simple reason that it is condensed, generally available and readable in small print. By experimenting I found that a 6 pt size will usually fill up the available space quite nicely.

Just using a small font doesn't help with the browsing. To that end, the roving eye must be given guidance and directions to keep it 'on track,' while the landscape should display enough markers for orientation. In the layout presented below, guidance is delivered by putting the listing in narrow, flush colums; album titles printed in boldface form landmarks in the overcrowded area.

To illustrate how individual tracks are typeset, the examples in Figure 2 show an excerpt of an imaginary listing at two different column widths.

The layout was inspired by the typesetting of the Key Index in *Mathematical Reviews*, which was used as an example of complex typesetting in the article *Breaking Paragraphs into Lines* [1] wherein Donald E. Knuth and Michael Plass explain the versatility of the glue-box-penalty model.

The anatomy of a single entry could be described as follows: The first line starts with the track number against the left margin; it is followed by some white space; then come the track title and artist; finally appears the track time flush against the right margin. Each entry is one paragraph. The space between the artist and the track time is filled up with a dotted line (technically called *leaders*). If there is not enough room for everything on one line, the title and artist are broken across several lines, where all lines but the first keep the same distance from the left margin so that they line out nicely with the start of the title in the first line, and all lines but the last have a ragged right margin. If there is no room left for the track time, it is placed on a line by itself with leaders in front aligning on the left with the title, as seen in (1).

Understanding how TeX is able to produce such intricate layouts requires some insight into the glue-box-penalty model. The most definitive source of information on this matter is the TeXbook, Chapter 14, but I will try to explain some of its particularities.

One of the aspects of TeX that really confused me as a novice user was the fact that it typesets paragraphs as a whole. I had grown accustomed to the linear typesetting behaviour of modern word processors: you type until the line is full and the word processor will insert a line break. At first, it will try to fill up the line by 'squeezing' it a little: the spaces are slightly narrowed, so more material fits on a line. But as you keep typing it must choose a breakpoint as close as possible to the end of the line. A clever word processor may reconsider the last breakpoint if it finds that it can hyphenate the word you are typing and the part before the hyphen will fit in the previous line; but even a clever word processor will not look further back than the last breakpoint. It doesn't care if a stretched line appears below a squeezed line, even if both lines could be improved by moving a small word from the end of the latter to the beginning of the former.

In contrast, TeX tries to balance an entire paragraph. The breakpoints are found by considering *all* candidates. The question it asks itself at every point is: "How bad would this paragraph be so far if I break it here? Or, how much do I need to stretch or shrink the line preceding it, and which previous breakpoint minimizes the total badness?" The notion that each potential breakpoint is linked to its previous breakpoint of minimal total badness introduces a "path of minimal badness" from the end of the paragraph right back to the beginning, traversing the 'optimal' breakpoints.

However clever, this approach has some repercussions. For instance, a novice user may be tempted to produce a single centered line in the middle of his paragraph for ad hoc displaying by typing something like

```
... by typing\\
{\centering some centered text\linebreak}
but to my dismay ...
```

but to his dismay, the display line
is hopelessly stretched.
That is because none of the parameters that affect the layout of paragraphs are used by TeX until it hits `\par`. At that moment, the paragraph builder springs into action, getting ready to build a perfectly balanced paragraph. The text has been reduced to a string of boxes, penalties and glues; the `\centering` declaration has already gone out of scope, the parameters it affected were reset when the `}` was read.

How, then, is it possible to have a paragraph that has the first line flush left, the middle lines indented, and the last line flush right? The answer is: a combination of standard TeXniques and some tricks.

The indentation of all but the first line is done with hanging indentation. Keeping the lines away from the right margin is done by setting `\rightskip`. This is a glue parameter that is added on the right of every line. The only real trick is to get the last line flush right. To do this, the `\rightskip` glue must be cancelled out. This can be done by putting *negative glue* next to it. The `\parfillskip` glue, that TeX inserts at the end of a paragraph, comes in handy here. This parameter usually has some stretch so the last line of the paragraph doesn't need to be full. If you have a hard time imagining what negative glue looks like, just consider the simple arithmetic it involves. People seem to have no problem whatsoever in everyday life understanding negative money.

Here is the list of parameters that are in effect when typesetting the entries.

```
\parindent 0pt
\leftskip 0em
\rightskip 2em plus 1.5em
\newlength{\threedigs}
\settowidth{\threedigs}{000\quad}
\hangafter 1
\hangindent \threedigs
\parfillskip -\rightskip
```

Then there is the matter of the leaders. There are two possible situations:

1. the track time goes on the same line as the last words of the track title and the leaders go between them;
2. there is not enough room left on the line, and the track time will have to go on a line of its own. The leaders will appear starting from the left edge (respecting the hanging indent).

The second case is really distinct because a linebreak will happen just before the leaders. Leaders, glue, and interword spaces are discarded at linebreaks. Normally this is a good thing, because you don't want spaces to appear at the beginning of a line. But in this case it is more aesthetic to have a line of leaders preceding the lonely track time. So in order to prevent the discarding, the following list of items is used:

- □ a penalty, indicating that it is possible (though undesirable) to break here;
- □ an empty `\hbox`, preventing the following leaders to be discarded if the above penalty should be chosen as a breakpoint;
- □ an infinite penalty, preventing a break *at* the following leaders (which would discard them);
- □ the leaders.

The result is that if a break happens at the first penalty, the empty box appears as the first item of the following line, in which case the leaders are safe.

The Index in *Mathematical Reviews* was even more elaborate, since it had a number of lines flush left, followed by leaders, followed by a number of lines flush right. See if you can figure out how to do *that*.

Translating the input

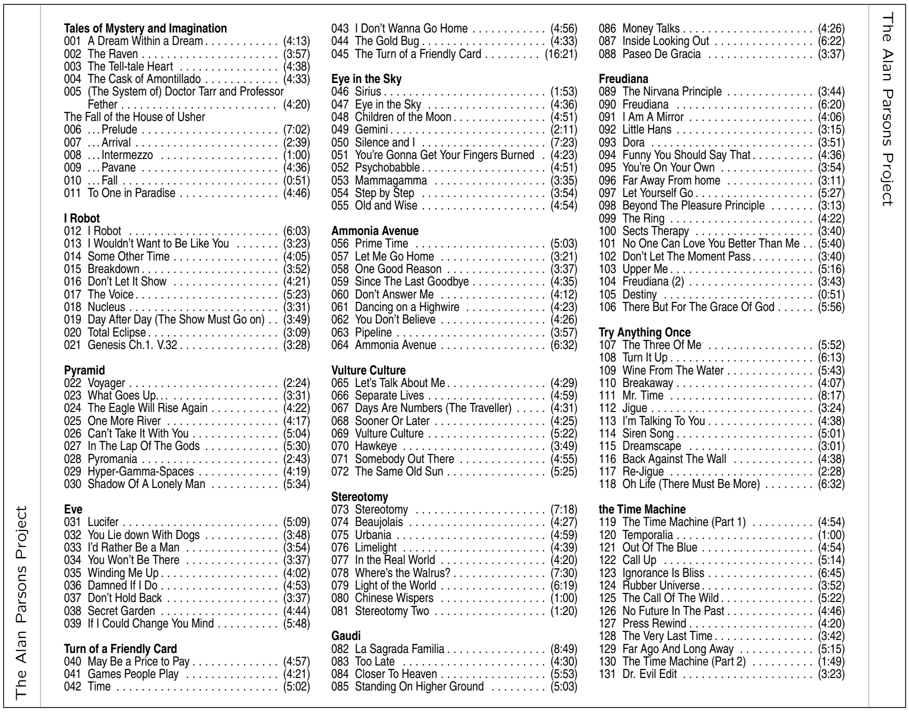
So now let's turn to the (human-readable) input file, and see how TEX is able to ingest it and produce the above layout.

The pivotal macro is called `\entry`, which has the following definition.

```
\def\entry#1 #2...(#3).{#1\track#2\tracktime(#3)\par}
```

What is immediately clear is that its ⟨parameter text⟩ is made to match the input format. The `\track` is 0.5 em of space. The `\tracktime` is the list of items that eventually result in the leaders, as was discussed in the previous section.

In order to make this control sequence magically appear before every line of the input, we set `\everypar` to `{\entry}`. This token list is inserted when TEX has begun a new paragraph. Since each `\entry` ends in a `\par`, every line of the input becomes an `\entry`. The only exceptions are the CD titles; since they should be exempted from becoming entries, the `\title` temporarily disables the `\everypar` and sets all the other parameters that were used for the list entries to



The Alan Parsons Project

Tales of Mystery and Imagination
001 A Dream Within a Dream (4:13)
002 The Raven (3:57)
003 The Tell-tale Heart (4:38)
004 The Cask of Amontillado (4:33)
005 (The System of) Doctor Tarr and Professor Fether (4:20)
The Fall of the House of Usher
006 …Prelude (7:02)
007 …Arrival (2:39)
008 …Intermezzo (1:00)
009 …Pavane (4:36)
010 …Fall (0:51)
011 To One in Paradise (4:46)

I Robot
012 I Robot (6:03)
013 I Wouldn't Want to Be Like You (3:23)
014 Some Other Time (4:05)
015 Breakdown (3:52)
016 Don't Let It Show (4:21)
017 The Voice (5:23)
018 Nucleus (3:31)
019 Day After Day (The Show Must Go on) (3:49)
020 Total Eclipse (3:09)
021 Genesis Ch.1. V.32 (3:28)

Pyramid
022 Voyager (2:24)
023 What Goes Up... (3:31)
024 The Eagle Will Rise Again (4:22)
025 One More River (4:17)
026 Can't Take It With You (5:04)
027 In The Lap Of The Gods (5:30)
028 Pyromania (2:43)
029 Hyper-Gamma-Spaces (4:19)
030 Shadow Of A Lonely Man (5:34)

Eve
031 Lucifer (5:09)
032 You Lie down With Dogs (3:48)
033 I'd Rather Be a Man (3:54)
034 You Won't Be There (3:37)
035 Winding Me Up (4:02)
036 Damned If I Do (4:53)
037 Don't Hold Back (3:37)
038 Secret Garden (4:44)
039 If I Could Change You Mind (5:48)

Turn of a Friendly Card
040 May Be a Price to Pay (4:57)
041 Games People Play (4:21)
042 Time (5:02)
043 I Don't Wanna Go Home (4:56)
044 The Gold Bug (4:33)
045 The Turn of a Friendly Card (16:21)

Eye in the Sky
046 Sirius (1:53)
047 Eye in the Sky (4:36)
048 Children of the Moon (4:51)
049 Gemini (2:11)
050 Silence and I (7:23)
051 You're Gonna Get Your Fingers Burned (4:23)
052 Psychobabble (4:51)
053 Mammagamma (3:35)
054 Step by Step (3:54)
055 Old and Wise (4:54)

Ammonia Avenue
056 Prime Time (5:03)
057 Let Me Go Home (3:21)
058 One Good Reason (3:37)
059 Since The Last Goodbye (4:35)
060 Don't Answer Me (4:12)
061 Dancing on a Highwire (4:23)
062 You Don't Believe (4:26)
063 Pipeline (3:57)
064 Ammonia Avenue (6:32)

Vulture Culture
065 Let's Talk About Me (4:29)
066 Separate Lives (4:59)
067 Days Are Numbers (The Traveller) (4:31)
068 Sooner Or Later (4:25)
069 Vulture Culture (5:22)
070 Hawkeye (3:49)
071 Somebody Out There (4:55)
072 The Same Old Sun (5:25)

Stereotomy
073 Stereotomy (7:18)
074 Beaujolais (4:27)
075 Urbania (4:59)
076 Limelight (4:39)
077 In the Real World (4:20)
078 Where's the Walrus? (7:30)
079 Light of the World (6:19)
080 Chinese Wispers (1:00)
081 Stereotomy Two (1:20)

Gaudi
082 La Sagrada Familia (8:49)
083 Too Late (4:30)
084 Closer To Heaven (5:53)
085 Standing On Higher Ground (5:03)
086 Money Talks (4:26)
087 Inside Looking Out (6:22)
088 Paseo De Gracia (3:37)

Freudiana
089 The Nirvana Principle (3:44)
090 Freudiana (6:20)
091 I Am A Mirror (4:06)
092 Little Hans (3:15)
093 Dora (3:51)
094 Funny You Should Say That (4:36)
095 You're On Your Own (3:54)
096 Far Away From home (3:11)
097 Let Yourself Go (5:27)
098 Beyond The Pleasure Principle (3:13)
099 The Ring (4:22)
100 Sects Therapy (3:40)
101 No One Can Love You Better Than Me (5:40)
102 Don't Let The Moment Pass (3:40)
103 Upper Me (5:16)
104 Freudiana (2) (3:43)
105 Destiny (0:51)
106 There But For The Grace Of God (5:56)

Try Anything Once
107 The Three Of Me (5:52)
108 Turn It Up (6:13)
109 Wine From The Water (5:43)
110 Breakaway (4:07)
111 Mr. Time (8:17)
112 Jigue (3:24)
113 I'm Talking To You (4:38)
114 Siren Song (5:01)
115 Dreamscape (3:01)
116 Back Against The Wall (4:38)
117 Re-Jigue (2:28)
118 Oh Life (There Must Be More) (6:32)

the Time Machine
119 The Time Machine (Part 1) (4:54)
120 Temporalia (1:00)
121 Out Of The Blue (4:54)
122 Call Up (5:14)
123 Ignorance Is Bliss (6:45)
124 Rubber Universe (3:52)
125 The Call Of The Wild (5:22)
126 No Future In The Past (4:46)
127 Press Rewind (4:20)
128 The Very Last Time (3:42)
129 Far Ago And Long Away (5:15)
130 The Time Machine (Part 2) (1:49)
131 Dr. Evil Edit (3:23)

The Alan Parsons Project

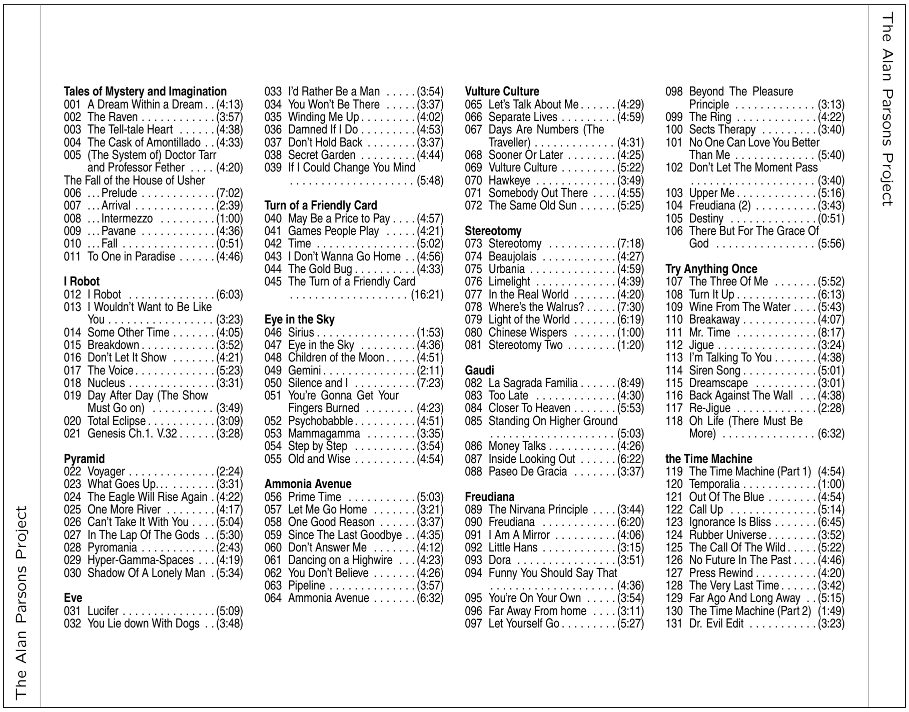
Figure 3. Example CD backside of my Alan Parsons Project collection.

more conventional values.

```
\def\title#1.{%
  \vskip\baselineskip % blank space
  \penalty-100%
  {%
    \everypar{}%
    \leavevmode
    \rightskip 0pt%
    \leftskip 0pt%
    \hangafter 0%
    \hangindent 0pt%
    \parfillskip \fill
    \fontseries{bc}\selectfont
    #1\par\nobreak
  }
}
```

The macro starts with a blank line. The `\penalty-100` indicates that this is a good point for breaking a column. The remainder of the declarations are grouped, so they only affect the current paragraph. The definition finishes on a `\nobreak` to prevent a 'widow' title at the bottom of the column.

The typeset results are displayed in Figure 3. The full LaTeX source and input data to produce this can be downloaded from `http://wwww.ntg.nl/maps/33/cdcases`.



The Alan Parsons Project

Tales of Mystery and Imagination
001 A Dream Within a Dream (4:13)
002 The Raven (3:57)
003 The Tell-tale Heart (4:38)
004 The Cask of Amontillado (4:33)
005 (The System of) Doctor Tarr and Professor Fether (4:20)
The Fall of the House of Usher
006 …Prelude (7:02)
007 …Arrival (2:39)
008 …Intermezzo (1:00)
009 …Pavane (4:36)
010 …Fall (0:51)
011 To One in Paradise (4:46)

I Robot
012 I Robot (6:03)
013 I Wouldn't Want to Be Like You (3:23)
014 Some Other Time (4:05)
015 Breakdown (3:52)
016 Don't Let It Show (4:21)
017 The Voice (5:23)
018 Nucleus (3:31)
019 Day After Day (The Show Must Go on) (3:49)
020 Total Eclipse (3:09)
021 Genesis Ch.1. V.32 (3:28)

Pyramid
022 Voyager (2:24)
023 What Goes Up… (3:31)
024 The Eagle Will Rise Again (4:22)
025 One More River (4:17)
026 Can't Take It With You (5:04)
027 In The Lap Of The Gods (5:30)
028 Pyromania (2:43)
029 Hyper-Gamma-Spaces (4:19)
030 Shadow Of A Lonely Man (5:34)

Eve
031 Lucifer (5:09)
032 You Lie down With Dogs (3:48)
033 I'd Rather Be a Man (3:54)
034 You Won't Be There (3:37)
035 Winding Me Up (4:02)
036 Damned If I Do (4:53)
037 Don't Hold Back (3:37)
038 Secret Garden (4:44)
039 If I Could Change You Mind (5:48)

Turn of a Friendly Card
040 May Be a Price to Pay (4:57)
041 Games People Play (4:21)
042 Time (5:02)
043 I Don't Wanna Go Home (4:56)
044 The Gold Bug (4:33)
045 The Turn of a Friendly Card (16:21)

Eye in the Sky
046 Sirius (1:53)
047 Eye in the Sky (4:36)
048 Children of the Moon (4:51)
049 Gemini (2:11)
050 Silence and I (7:23)
051 You're Gonna Get Your Fingers Burned (4:23)
052 Psychobabble (4:51)
053 Mammagamma (3:35)
054 Step by Step (3:54)
055 Old and Wise (4:54)

Ammonia Avenue
056 Prime Time (5:03)
057 Let Me Go Home (3:21)
058 One Good Reason (3:37)
059 Since The Last Goodbye (4:35)
060 Don't Answer Me (4:12)
061 Dancing on a Highwire (4:23)
062 You Don't Believe (4:26)
063 Pipeline (3:57)
064 Ammonia Avenue (6:32)

Vulture Culture
065 Let's Talk About Me (4:29)
066 Separate Lives (4:59)
067 Days Are Numbers (The Traveller) (4:31)
068 Sooner Or Later (4:25)
069 Vulture Culture (5:22)
070 Hawkeye (3:49)
071 Somebody Out There (4:55)
072 The Same Old Sun (5:25)

Stereotomy
073 Stereotomy (7:18)
074 Beaujolais (4:27)
075 Urbania (4:59)
076 Limelight (4:39)
077 In the Real World (4:20)
078 Where's the Walrus? (7:30)
079 Light of the World (6:19)
080 Chinese Wispers (1:00)
081 Stereotomy Two (1:20)

Gaudi
082 La Sagrada Familia (8:49)
083 Too Late (4:30)
084 Closer To Heaven (5:53)
085 Standing On Higher Ground (5:03)
086 Money Talks (4:26)
087 Inside Looking Out (6:22)
088 Paseo De Gracia (3:37)

Freudiana
089 The Nirvana Principle (3:44)
090 Freudiana (6:20)
091 I Am A Mirror (4:06)
092 Little Hans (3:15)
093 Dora (3:51)
094 Funny You Should Say That (4:36)
095 You're On Your Own (3:54)
096 Far Away From home (3:11)
097 Let Yourself Go (5:27)
098 Beyond The Pleasure Principle (3:13)
099 The Ring (4:22)
100 Sects Therapy (3:40)
101 No One Can Love You Better Than Me (5:40)
102 Don't Let The Moment Pass (3:40)
103 Upper Me (5:16)
104 Freudiana (2) (3:43)
105 Destiny (0:51)
106 There But For The Grace Of God (5:56)

Try Anything Once
107 The Three Of Me (5:52)
108 Turn It Up (6:13)
109 Wine From The Water (5:43)
110 Breakaway (4:07)
111 Mr. Time (8:17)
112 Jigue (3:24)
113 I'm Talking To You (4:38)
114 Siren Song (5:01)
115 Dreamscape (3:01)
116 Back Against The Wall (4:38)
117 Re-Jigue (2:28)
118 Oh Life (There Must Be More) (6:32)

the Time Machine
119 The Time Machine (Part 1) (4:54)
120 Temporalia (1:00)
121 Out Of The Blue (4:54)
122 Call Up (5:14)
123 Ignorance Is Bliss (6:45)
124 Rubber Universe (3:52)
125 The Call Of The Wild (5:22)
126 No Future In The Past (4:46)
127 Press Rewind (4:20)
128 The Very Last Time (3:42)
129 Far Ago And Long Away (5:15)
130 The Time Machine (Part 2) (1:49)
131 Dr. Evil Edit (3:23)

The Alan Parsons Project

Figure 4. CD backside in four columns.

Exploring alternatives

Looking at the examples of Figure 2, one thing that really stands out is what a difference a slightly wider or narrower column can make. For example, (1) might be used for a four-column, and (2) for a three-column layout. Although four columns have 1/3 more lines, (1) requires 2/5 more lines than (2). The optimal number of columns can only be found by experimenting, although as a general rule wider entries need wider columns.

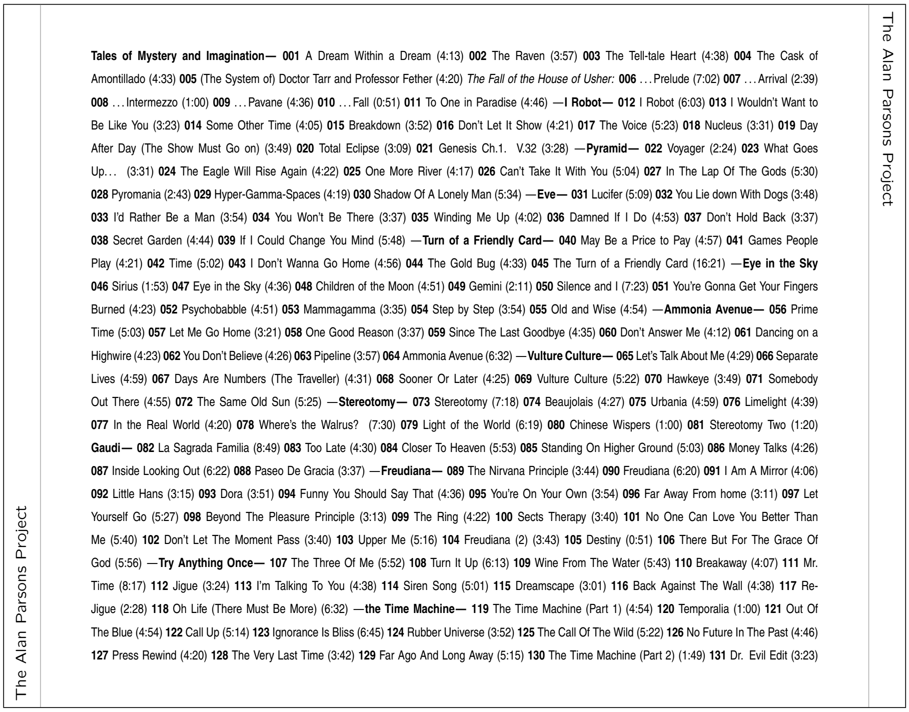
To show the difference between three and four columns, Figure 4 shows the same listing as Figure 3, only in a four-column setting. Here, the space advantage goes to the four column layout. The good looks were somewhat compromised, though, because 14 entries needed to be spread over multiple lines against only one in the three column version. The overall feel is more staggered. I will spare you the results if the number of columns is set to *five*. But there is something else that can be done.

There is a remarkable parallel between the format of the input and the resulting index. But although this was by design, the similarity has no deeper meaning. There is, for instance, no relation between the three dots that appear in the input and the leaders that are inserted in the final typesetting. Even the parentheses around the track time are part of the macro template, so that we could format this entirely differently if we wanted.

This led me to think of an entirely different approach to the problem of a tight, but browsable layout. Why should the solution have to have columns? Why could guidance and directions not be given in another form? The human eye may have some surprises for us; I will present an alternative that will have a *horizontal* accent rather than *vertical*. Whether this is better, or more beautiful than the original I leave up to the reader to decide.

The layout in Figure 5 shows the results; we'll get to the TEXnicalities below.

One thing that stands out is the overall uniformness; there are no unsightly gaps.



The Alan Parsons Project

Tales of Mystery and Imagination— 001 A Dream Within a Dream (4:13) **002** The Raven (3:57) **003** The Tell-tale Heart (4:38) **004** The Cask of Amontillado (4:33) **005** (The System of) Doctor Tarr and Professor Fether (4:20) *The Fall of the House of Usher:* **006** . . . Prelude (7:02) **007** . . . Arrival (2:39) **008** . . . Intermezzo (1:00) **009** . . . Pavane (4:36) **010** . . . Fall (0:51) **011** To One in Paradise (4:46) —**I Robot— 012** I Robot (6:03) **013** I Wouldn't Want to Be Like You (3:23) **014** Some Other Time (4:05) **015** Breakdown (3:52) **016** Don't Let It Show (4:21) **017** The Voice (5:23) **018** Nucleus (3:31) **019** Day After Day (The Show Must Go on) (3:49) **020** Total Eclipse (3:09) **021** Genesis Ch.1. V.32 (3:28) —**Pyramid— 022** Voyager (2:24) **023** What Goes Up. . . (3:31) **024** The Eagle Will Rise Again (4:22) **025** One More River (4:17) **026** Can't Take It With You (5:04) **027** In The Lap Of The Gods (5:30) **028** Pyromania (2:43) **029** Hyper-Gamma-Spaces (4:19) **030** Shadow Of A Lonely Man (5:34) —**Eve— 031** Lucifer (5:09) **032** You Lie down With Dogs (3:48) **033** I'd Rather Be a Man (3:54) **034** You Won't Be There (3:37) **035** Winding Me Up (4:02) **036** Damned If I Do (4:53) **037** Don't Hold Back (3:37) **038** Secret Garden (4:44) **039** If I Could Change You Mind (5:48) —**Turn of a Friendly Card— 040** May Be a Price to Pay (4:57) **041** Games People Play (4:21) **042** Time (5:02) **043** I Don't Wanna Go Home (4:56) **044** The Gold Bug (4:33) **045** The Turn of a Friendly Card (16:21) —**Eye in the Sky 046** Sirius (1:53) **047** Eye in the Sky (4:36) **048** Children of the Moon (4:51) **049** Gemini (2:11) **050** Silence and I (7:23) **051** You're Gonna Get Your Fingers Burned (4:23) **052** Psychobabble (4:51) **053** Mammagamma (3:35) **054** Step by Step (3:54) **055** Old and Wise (4:54) —**Ammonia Avenue— 056** Prime Time (5:03) **057** Let Me Go Home (3:21) **058** One Good Reason (3:37) **059** Since The Last Goodbye (4:35) **060** Don't Answer Me (4:12) **061** Dancing on a Highwire (4:23) **062** You Don't Believe (4:26) **063** Pipeline (3:57) **064** Ammonia Avenue (6:32) —**Vulture Culture— 065** Let's Talk About Me (4:29) **066** Separate Lives (4:59) **067** Days Are Numbers (The Traveller) (4:31) **068** Sooner Or Later (4:25) **069** Vulture Culture (5:22) **070** Hawkeye (3:49) **071** Somebody Out There (4:55) **072** The Same Old Sun (5:25) —**Stereotomy— 073** Stereotomy (7:18) **074** Beaujolais (4:27) **075** Urbania (4:59) **076** Limelight (4:39) **077** In the Real World (4:20) **078** Where's the Walrus? (7:30) **079** Light of the World (6:19) **080** Chinese Wispers (1:00) **081** Stereotomy Two (1:20) **Gaudi— 082** La Sagrada Familia (8:49) **083** Too Late (4:30) **084** Closer To Heaven (5:53) **085** Standing On Higher Ground (5:03) **086** Money Talks (4:26) **087** Inside Looking Out (6:22) **088** Paseo De Gracia (3:37) —**Freudiana— 089** The Nirvana Principle (3:44) **090** Freudiana (6:20) **091** I Am A Mirror (4:06) **092** Little Hans (3:15) **093** Dora (3:51) **094** Funny You Should Say That (4:36) **095** You're On Your Own (3:54) **096** Far Away From home (3:11) **097** Let Yourself Go (5:27) **098** Beyond The Pleasure Principle (3:13) **099** The Ring (4:22) **100** Sects Therapy (3:40) **101** No One Can Love You Better Than Me (5:40) **102** Don't Let The Moment Pass (3:40) **103** Upper Me (5:16) **104** Freudiana (2) (3:43) **105** Destiny (0:51) **106** There But For The Grace Of God (5:56) —**Try Anything Once— 107** The Three Of Me (5:52) **108** Turn It Up (6:13) **109** Wine From The Water (5:43) **110** Breakaway (4:07) **111** Mr. Time (8:17) **112** Jigue (3:24) **113** I'm Talking To You (4:38) **114** Siren Song (5:01) **115** Dreamscape (3:01) **116** Back Against The Wall (4:38) **117** Re-Jigue (2:28) **118** Oh Life (There Must Be More) (6:32) —**the Time Machine— 119** The Time Machine (Part 1) (4:54) **120** Temporalia (1:00) **121** Out Of The Blue (4:54) **122** Call Up (5:14) **123** Ignorance Is Bliss (6:45) **124** Rubber Universe (3:52) **125** The Call Of The Wild (5:22) **126** No Future In The Past (4:46) **127** Press Rewind (4:20) **128** The Very Last Time (3:42) **129** Far Ago And Long Away (5:15) **130** The Time Machine (Part 2) (1:49) **131** Dr. Evil Edit (3:23)

The Alan Parsons Project

Figure 5. The same listing as before, only now *inline*.

The listing occupies an entire rectangular area. Since there is no space going into any leaders, this layout is definitely the most economic. This allows for a much larger interline space, which gives more 'air' and some welcome guidance along the very long lines.

The track numbers are now put in bold, which makes them stand out, strewn across the page like spots on a Dalmatian. The larger landmarks are formed by the boldface album titles, which are adorned with em-dashes for extra horizontal accenting.

So there it is: guidance and directions for the browsing eye. I would dare to stipulate that looking up the track number for a given song will, on average, take no longer in this version than in the column layout. I admit that counting the number of tracks of an album or calculating the total time of an album probably takes a little longer.

What is remarkable about this layout is that *it was generated from exactly the same input file as the original!* It just took a little more trickery.

In contrast with the column layout, where each entry was a paragraph, the inline listing is put in a single paragraph. We've seen the use of `\parfillskip` earlier, and here it is set to `0pt` to make the listing come out exactly rectangular. With a long listing like this, TEX will have no trouble finding suitable breakpoints to make it happen. Here the paragraph balancing is really showing off.

The listing is placed in a `\parbox` with a predetermined width and height, so the margins around the text appear nicely uniform. The interline space is flexible, so the text will stretch to fill the desired height.

```
\baselineskip 10pt plus 10pt
```

Attentive readers will notice how the em-dashes disappear at the beginning and end of lines. If you remember the discussion about the leaders and how to protect them from being discarded, you'll probably have guessed (and rightly so) that the em-dashes are in fact leaders.

```
\def\leaddash{\cleaders\hbox to 1em{\hss---\hss}\hskip 1em}
```

A stupid kludge was needed to make the dash disappear for the first title; since this does not appear at a line break, no glue is discarded. I had to insert a line break right at the beginning, and back the entire text up vertically to compensate for the blank line.

```
\vskip -6pt\mbox{}\\
```

The really tricky part was to read the input. I could not use \everypar again, since there is only one large paragraph. So I made the newline character an active character, to insert the \entry macro for every line.

```
\let\par\entry
\obeylines
```

That left me with handling the lines beginning with \title. In the column layout, the \title was read before the \everypar tokens were inserted, so they could be temporarily turned off. here, the newline character is seen *before* \title, so the \entry macro has to do some looking ahead.

```
\def\entry{\space\futurelet\next\bentry}
\def\bentry{\ifcat\noexpand\next0\tentry\fi}
```

The \futurelet allows the looking ahead of one token. So \next is set to be either \title or the first digit of the track number. Control is then passed to \bentry. This macro compares the category code of \next with that of a digit. The \noexpand is necessary because \ifcat would otherwise expand \next if it were \title. Digits have category code 12, while a control sequence has category code 16. If a digit is detected, \tentry is inserted which handles the entry like before.

```
\def\tentry#1 #2...(#3).{\tracknr{#1}\track{#2}\tracktime(#3)}
```

Otherwise nothing is inserted and the \title handles the line.

```
\def\title#1.{\leaddash{\fontseries{bc}\selectfont#1\leaddash}}
```

As can be seen, the inline lay-out has very little formatting work to do.

```
\def\tracknr#1{{\fontseries{bc}\selectfont #1}}
\def\track#1{\nobreak\space #1}}
\def\tracktime(#1){\nobreak\space(#1)}
```

The \nobreaks prevent breaking a line between the track number and title, or between the title and the time.

The full LaTEX source and input data to produce this layout can be downloaded from http://wwww.ntg.nl/maps/33/cdcases.

In conclusion, creating a good looking CD case layout is best left to TEX. The examples shown could not have been made with any common word processor. Furthermore, there is full separation of content and layout, which is the boon of today's information gurus. Whether you like columns or in-line, the input file is the same.

Postscriptum

I learned a lot about TEX during the writing of this article. I designed the layout several years ago and didn't bother to make it too clean. Writing about it in this detail forced me to clean up the code and verify everything for correctness (which never turned out to be the case *entirely*). Eventually I redid a lot of stuff. I wanted to show the input parsing strength by having an input format with hardly a control sequence in it (which the original had). With some extra effort, the last control sequences (\titles, mostly) could disappear as well. It is just a matter of comparing the catcodes, and making sure the title lines start with a letter instead of a number.

I have tried very hard to make sure the examples work as described; for those who are genuinely interested I will make the scripts and test files available on the Internet.

The idea to try an alternative, inline layout came as an afterthought, while I was already halfway through the article. Coding, experimenting and documenting was done in the last minute; I don't actually have any experience with the usability of the result. If anyone tries it out, I'm curious about what they think.

I owe many thanks to Wybo Dekker, who came up with the Ruby script mp3totex as an elegant replacement for the rather kludgy Python program I originally used. It can easily be enhanced to support larger collections of files and different output formats.

The mp3totex program, the macros and finished layouts in this article were for a CD collection of one band. The examples from Figure 2 are more suited for CDs with *various artists*, where it makes sense to include the artist name with the track. This requires some modifications to script and macros.

I created the layouts some years ago, and time is catching up with old technology. People are carrying around MP3 players the size of a deck of cards and ten times the storage of a CD. Already, DVDs are replacing CDs as the default optical medium.

I have no hope of ever creating a readable layout that will fit thousands of titles on the back of an iPod, but a DVD jewel case booklet sounds just about doable.

References

[1] D. E. Knuth and M. F. Plass. Breaking paragraphs into lines. In *Digital Typography*, volume 78 of *CSLI Lecture Notes*, chapter 3, pages 67–155. CSLI Publications, Stanford, California, 1998. Originally published in Software—Practice and Experience **11** (1981), 1119–1184.

Dennis van Dok
dvandok@quicknet.nl

Font installation the shallow way

Abstract
For one-off projects, you can cut corners with font installation and end up with a more manageable set of files and a cleaner TeX installation. This article shows how and why.

Keywords
Font installation, afm2pl, afm2tfm, TrueType, pdftex, mapfiles

If you are putting together a flyer or invitation or book cover, then it would be nice if you could without too much trouble test a batch of fonts from your CorelDRAW- or Illustrator cd or your Windows font directory, without polluting your TeX installation with a lot of stuff you are never going to use again.

This article takes you through the steps needed to use one or more fonts in one particular document. We won't really install the fonts; we just generate the files that TeX needs and leave them where TeX will find them, *i.e.* in the working directory. This makes it easy to take the project to another system, and easy to clean things up.

We will primarily use afm2pl to generate `.tfm` (TeX Font Metric) files. Later on, we show the steps required for afm2tfm. Both programs are simpler and much faster to use than the usual choice, fontinst. They create few intermediate or unnecessary files and do their job without virtual fonts. Virtual fonts and fontinst have their place, but sometimes there is no good reason to put up with the inevitable mess.

afm2tfm is available on all major free TeX implementations. afm2pl is part of current TeX Live distributions. Note that these programs are needed only to create the necessary font support files for TeX; once these files have been created, they can be used on any other system, whether or not it contains afm2pl or afm2tfm.

An example

We use a decorative script font Pepita that Adobe bundles or used to bundle with some of its software.

pdftex will need the actual font file `epscr___.pfb`, its TeX font metrics file `epscr7t.tfm` and a mapfile containing an entry relating the two. First, we copy not only `epsrc___.pfb` but also `epsrc___.afm` to the working directory. We need the latter file to generate the `.tfm` file. Next, we enter the following commands on a command line:

```
afm2pl -p ot1 epscr___.afm epscr7t.pl
pltotf epscr7t
```

The extensions `.afm` and `.pl` are optional. The first command converts the `.afm` file to an (almost) human-readable text version of the desired `.tfm` file. The second command creates the more compact binary version.

Before we can use this font, we must LaTeX tell about it. We do this with a font family definition file `ot1myfontfam.fd`:

```
\ProvidesFile{ot1myfontfam.fd}
\DeclareFontFamily{OT1}{myfontfam}{}
\DeclareFontShape{OT1}{myfontfam}{m}{n}{
  <-> epscr7t }{}
```

The prefix `ot1` indicates the encoding, which tells which characters occur at what positions. The next section will say more about encodings. The parameters to `\DeclareFontShape` are successively encoding, family name, weight (*e.g.* bold), shape, font file (without extension) and special options. You can normally leave this last parameter empty. With just one family member, we are not fussy about font characteristics and just pick defaults. We also leave this file in the working directory.

This is the code of our first testfile `exabasic.tex`, which uses this font:

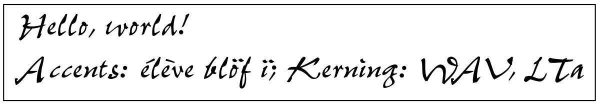
```
\documentclass{article}
\pagestyle{empty}
\pdfmapfile{=epscr7t.map}

\newcommand{\fancyfont}%
  {\fontfamily{myfontfam}\selectfont}

\begin{document}
\fancyfont
Hello, world!

Accents: \'el\`eve bl\"of \"i;
Kerning: WAV, LTa
\end{document}
```

The `\pdfmapfile` command causes pdflatex to read the file `epscr7t.map` which tells pdftex how to get the font into the output file. The prepended '=' tells pdftex that it should read epscr7t.map *in addition to,* not instead of the default mapfile, and that in case of a conflict epscr7t.map wins.

Now we are ready to compile exabasic.tex:

```
pdflatex exabasic
```

This is the result:



Hello, world!

Accents: élève blöf ï; Kerning: WAV, LTa

Encodings

We already made brief mention of encodings. Now is the time to dig a little deeper, because it is a topic that can easily trip you up.

An encoding defines what character corresponds to which number. Only numbers between 0 and 255 are allowed. A .tfm file associates character metrics directly with character positions and doesn't know what position represents what character. TEX simply makes assumptions about this correspondence or encoding, and if you disagree with those assumptions then you need to load some macro package or other to tell TEX otherwise.

We hope that mainstream TeX will eventually move to Unicode, which is a comprehensive encoding of all conceivable characters, including far-eastern alphabets and mathematical symbols. When that happens, we can forget about encodings and also do away with many applications of virtual fonts. There are already some Unicode-based variants of TeX.[1]

For a PostScript .pfb- or .pfa font, character metrics are stored in a separate .afm file. These metrics are associated with characters, not with character positions. Therefore you should specify an encoding to afm2pl or afm2tfm[2]. The same encoding must also be specified in the mapfile entry. A PostScript font usually has more characters than fit into a single encoding.

A parameter '-p texnansi' or '-p texnansi.enc' means that the encoding should be read from a file texnansi.enc. This encoding probably has a different internal name.

OT1 encoding. If you don't tell TEX otherwise, it assumes that you use OT1 encoding. This encoding uses only 128 of the 256 available slots. TEX creates missing accented characters from an unaccented base character and a separate accent character. Unfortunately, this interferes with hyphenation. Apart from this, the OT1 encoding has various other oddities, and is best avoided. OT1-encoded fonts often have a TEX name ending in 7t[3]. Note that ot1.enc comes with afm2pl and is probably not available if you don't have afm2pl on your system.

T1 encoding. T1 is the successor to OT1. It uses all available slots, and has lots of accented characters, also for Eastern European languages. Because the T1 encoding left no room for typographic symbols such as '‰' or '©' or 'ƒ' you will need to get those from a second encoding of the same font. This second encoding is called TS1 or 'text companion'.

For most traditional PostScript fonts, some of the accented characters in the T1 encoding aren't actually present and must be created with virtual font technology from a base character and an accent. Since it doesn't have to be done by TEX itself, this is no obstacle to hyphenation.

Although you can tell afm2pl to use T1 encoding, it can't create composite characters, and such composite characters will be missing unless they are already present in the original font.

T1-encoded fonts often have a TEX name ending in 8t.

Texnansi encoding. Texnansi has been introduced by Y&Y, the now-defunct company behind Y&YTEX, dviwindow and dvipsone. It combines a good selection of both accented letters and typographic symbols, and normally contains everything you need in a single encoding, at least for Western European languages. Texnansi-encoded fonts often have a name ending in 8y.

The package texnansi selects the *texnansi* encoding and contains some additional code to smooth out incompatibilities with T1 and OT1.

A texnansi example. For this example, we choose Augie, a handwriting font from TeX Live. These are the commands for generating the .tfm and .map files:

```
afm2pl -p texnansi augie___.afm augie8y.pl
pltotf augie8y
```

This is ly1augie.fd (notice the ly1 prefix):

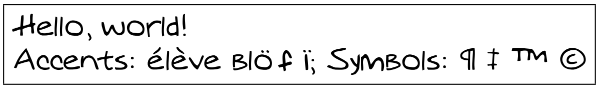
```
\ProvidesFile{ly1augie.fd}
\DeclareFontFamily{LY1}{augie}{}
\DeclareFontShape{LY1}{augie}{m}{n}{
  <-> augie8y }{}
```

This is the LateX code:

```
\documentclass{article}
\usepackage{texnansi}
\pagestyle{empty}
\pdfmapfile{=augie8y.map}

\newcommand{\fancyfont}%
  {\fontfamily{augie}\selectfont}
```

```
\begin{document}
\fancyfont
Hello, world!

Accents: \'el\`eve bl\"of \"i;
Symbols:
\textparagraph{} \textdaggerdbl{}
\texttrademark{} \textcopyright
\end{document}
```

and this is the result. Notice the extra symbols. These are absent from the T1 encoding and would have required a text companion font.



Hello, world!
Accents: élève blöf ï; Symbols: ¶ ‡ ™ ©

TrueType

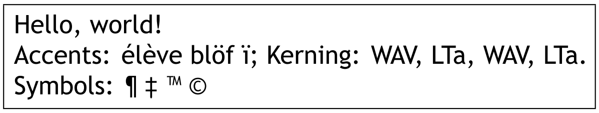
Another scalable font format is TrueType, which is supported by pdftex but currently not by dvips. Font metrics are stored in the font file itself. Using TrueType is somewhat more work; the following commands are required to import a TrueType font such as Trebuchet:

```
ttf2afm trebuc.ttf >trebuc.afm
afm2pl -p texnansi trebuc trebuc8y
pltotf trebuc8y
<edit mapfile to replace .pfb with .ttf>
```

ttf2afm extracts the metric information from the .ttf file.[4]

afm2pl has no way of knowing that the .afm describes a TrueType font, and guesses that the actual fontfile is trebuc.pfb. Therefore you have to fix the mapfile manually in an editor.

We leave it as an exercise for the reader to write the .fd file and LaTeX source for the following example:



Hello, world!
Accents: élève blöf ï; Kerning: WAV, LTa, WAV, LTa.
Symbols: ¶ ‡ ™ ©

Font-based uppercasing and letterspacing

afm2pl comes with an uppercased version *texnanuc* of texnansi. Uppercasing, *e.g.* in headings, works best in combination with letterspacing. For this, afm2pl has a parameter '-m'.

Warning. afm2pl implements letterspacing with kerns. Unfortunately, the .tfm format can contain only a limited number of kerns. If there are too many in the .pl file then all kerns and ligatures will be dropped from the generated .tfm file! So use this feature with care. fontinst implements letterspacing by adding sidebearings via virtual fonts, and doesn't suffer from this limitation.

We can create a letterspaced, uppercased version of Trebuchet with the following commands:

```
ttf2afm trebuc.ttf >trebuc.afm
afm2pl -p texnanuc -m 100 trebuc trebucupp8y
pltotf trebucupp8y
<edit mapfile to replace .pfb with .ttf>
```

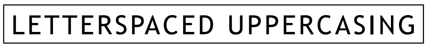
A fontfamily and fontshape declaration might look as follows:

```
\ProvidesFile{ly1trebuc.fd}
\DeclareFontFamily{LY1}{trebuc}{}
\DeclareFontShape{LY1}{trebuc}{m}{upp}{
  <-> trebucupp8y }{}
```

The fontshape upp for uppercasing is not an official LaTeX shape but that doesn't seem to matter. You can use the font as follows:

```
\documentclass{article}
\usepackage{texnansi}
\pagestyle{empty}
\pdfmapfile{=trebucupp8y.map}

\begin{document}
\fontfamily{trebuc}\fontshape{upp}\selectfont
Letterspaced uppercasing
\end{document}
```

and this is the result:



LETTERSPACED UPPERCASING

A font family

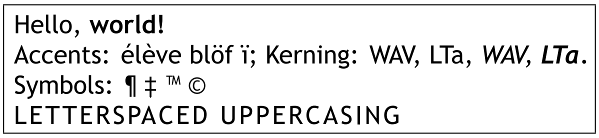
The next example uses a real font family, consisting of the usual four family members plus our letterspaced font. So we will need not only trebuc.ttf, as in the previous example, but also trebucbd.ttf, trebucit.ttf, and trebucbi.ttf. For each of these we'll have to run the ttf2afm – afm2pl – pltotf sequence, and we'll have to edit each of the generated map files, or create a combined mapfile.
This is its code of the .fd file:

```
\ProvidesFile{ly1trebuc.fd}
\DeclareFontFamily{LY1}{trebuc}{}
\DeclareFontShape{LY1}{trebuc}{bx}{n}{
  <-> trebucbd8y }{}
\DeclareFontShape{LY1}{trebuc}{m}{n}{
  <-> trebuc8y }{}
\DeclareFontShape{LY1}{trebuc}{bx}{it}{
  <-> trebucbi8y }{}
\DeclareFontShape{LY1}{trebuc}{m}{it}{
  <-> trebucit8y }{}
\DeclareFontShape{LY1}{trebuc}{m}{upp}{
  <-> trebucupp8y }{}
```

And this is the LaTeX code using it:

```
\documentclass{article}
\usepackage{texnansi}
\pagestyle{empty}
% better combine these mapfiles!
\pdfmapfile{=trebuc8y.map}
\pdfmapfile{=trebucbd8y.map}
\pdfmapfile{=trebucit8y.map}
\pdfmapfile{=trebucbi8y.map}
\pdfmapfile{=trebucupp8y.map}

\begin{document}
\fontfamily{trebuc}\selectfont
Hello, \textbf{world!}

Accents: \'el\`eve bl\"of \"i;
Kerning: WAV, LTa, \textit{WAV, \textbf{LTa.}}

Symbols:
\textparagraph{} \textdaggerdbl{}
\texttrademark{} \textcopyright

\fontshape{upp}\selectfont
Letterspaced uppercasing
\end{document}
```

This is the result:



Hello, **world!**
Accents: élève blöf ï; Kerning: WAV, LTa, *WAV, **LTa***.
Symbols: ¶ ‡ ™ ©
LETTERSPACED UPPERCASING

Using dvips

If you go the dvips route, then you cannot use the \pdfmapfile macro. Instead, you have to enter additional mapfiles on the command line:

```
dvips -u +mapfile dvifile
```

The prefix + to the mapfile parameter is analogous to the = prefix for the \pdfmapfile macro: it tells dvips to use the named mapfile *in addition to* the default one.

Using afm2tfm

The intention of afm2tfm is not to create fonts which are used directly by TeX. Instead, they serve as a basis for virtual fonts, *i.e.* recipes to compose fonts from other fonts. But it is not too difficult to subvert this intention:

```
afm2tfm epscr___ -T texnansi \
  -v indirect.vpl direct.tfm >direct.map
#rm direct.tfm
vptovf indirect.vpl
rm indirect.vf
<edit direct.map>
```

Note that the .afm filename comes *before* the options.

vptovf generates two files from indirect.vpl: indirect.vf and indirect.tfm.

You should remove indirect.vf, otherwise the dvi driver or pdftex would think that indirect is a virtual font.

Normally, you would also remove direct.tfm, but I keep it to show you the difference with indirect.tfm.

Mapfile information is written to standard output, which therefore had to be redirected, as shown above. It contains the following string:

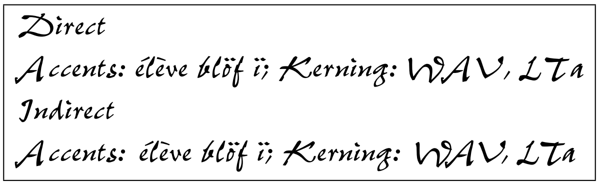
```
direct PepitaMT
  " TeXnANSIEncoding ReEncodeFont " <texnansi
```

(everything on one line). This has to be changed into:

```
indirect PepitaMT
  " TeXnANSIEncoding ReEncodeFont "
  <texnansi.enc <epscr___.pfb
```

(one line).

The example below displays differences in spacing between the two. *Note.* This is not an example for copying.



Direct
Accents: élève blöf ï; Kerning: WAV, LTa
Indirect
Accents: élève blöf ï; Kerning: WAV, LTa

Other options of afm2pl and afm2tfm

With both programs you can artificially slant, narrow and widen a font. afm2tfm can also generate artificial smallcaps. Such manipulated fonts rarely look good, though.

afm2pl also has some options for manipulating the ligkern table and for setting spacing parameters. For casual use, you don't bother with these.

OpenType

We are seeing more and more OpenType fonts, which are Unicode-based. These consist of either PostScript/Type 1 or TrueType outlines inside a TrueType wrapper. OpenType fonts may contain huge charactersets, sometimes including smallcaps and oldstyle figures.

OpenType fonts with Type 1 outlines, which have .otf extension, can be converted with otftotfm, part of Eddie Kohler's LCDF Typetools and included in TeX Live.

OpenType fonts with TrueType outlines have an extension .ttf and can be treated just like TrueType fonts.

Ad hoc or generic solutions?

Various people have written scripts to automate font installation. ConTeXt users will be familiar with texfont, which, by the way, has an option to use afm2pl instead of afm2tfm.

Each example took several commands on a command line. So why not a script?

I don't install fonts all that often. I like to decide case by case how to do it: what tools to use, what variants to generate, where to install or not install, how to name the fonts…

Under these circumstances, the simplest and best solution is to either do it by hand, or to write little ad-hoc scripts and keep them with the project.

Siep Kroonenberg
siepo@cybercomm.nl

Notes

1. Omega and its offshoot Aleph are Unicode-based. Users of Mac OS X may be interested in XeTeX (http://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&item_id=xetex), which is built on top of a regular TEX installation and lets you use Mac OS X unicode fonts directly with TEX.

2. If you don't specify an encoding, then you get the encoding from the .afm file, which is almost certainly not what you want.

3. For afm2pl and afm2tfm, font names have no particular meaning. This is one more difference with fontinst. I add encoding postfixes such as 7t and 8y to font names just as reminders to myself.

4. This will result in an empty encoding, unless you specify an encoding parameter. But we are going to ignore the encoding in the .afm anyhow.

Installing Expert Fonts: Minion Pro

Abstract
Installing fonts for ConTEXt can be intimidating business. In this issue we take on a real monster: a collection of Adobe Minion Pro expert fonts. We hope our installation of this collection will provide an illustrative example for ConTEXt users, and help to ease the pain of installing new fonts (if you can install Minion Pro, Myriad Pro and Poetica, you can install just about anything!).

Introduction

Fonts can be a messy business in TEX (and, by extension, ConTEXt), and it's easy to get intimidated. One reason for this is TEX's flexibility; TEX allows you to create very sophisticated ways to take advantage of a font and to create, from one or more given font families, typeface collections tailored to your needs. Another reason is a (hopefully temporary) lack of standardization of map and encoding files between pdfe-TEX, dvips, and dvipdfmx. This second reason is not really a ConTEXt problem per se, though it certainly affects getting fonts working in ConTEXt.

Furthermore, ConTEXt handles fonts and font families by means of *typescripts*; these can be a bit disorienting to someone coming from LaTEX and the New Font Selection Scheme (NFSS). On the other hand, after initial hesitation (having myself migrated from the LaTEX world), I have concluded that the typescript approach is much more powerful and transparent than NFSS.

For a present book project, I decided to use a very complicated set of fonts from Adobe: Minion Pro (roman or serif), Myriad Pro (sans serif) and Poetica (calligraphy); all by Robert Slimbach. This set also includes a number of expert fonts with non-standard encodings. Together – and aside from mathematics – this set can provide a very nice alternative to the Computer/Latin Modern family, and one particularly suited for the humanities. These fonts also provide some of the few really excellent examples of *multiple master* (MM) technology, by Adobe. The promise of MM font technology was to provide a means of creating a series of finely optically scaled styles and alternative of a font from a single font file.[1]

On the other hand, despite its promise the system was never widely used and Adobe apparently no longer fully supports it.

In the present experiment we will focus on installing Minion Pro. I will not attempt to fine tune the weights; I will just use the defaults (mostly 2 weights per variation, plus a semibold style).[2] There is also a Minion Pro Opticals family, which I received while writing this issue. Although this tutorial is based on the older Minion Pro familiar to advanced LaTEX users, Appendix 1 explains how to set up Minion Pro Opticals. It should be easy to follow for anyone who has read the earlier sections, and provides a nice example of a truly advanced typescript.

Our work may be divided into three parts:

1. preparing the raw fonts;
2. installing the fonts; and
3. configuring typescripts and map files to use the fonts.

Ok, let's get to work!

Preparing the Fonts

Fonts generally will come in one of three forms: Type1 (`*.pfb`), TrueType (`*.ttf`), and OpenType (`*.otf`). TeX was generally restricted to Type1 fonts until recently. pdfe-TeX supports the other two to some degree. dvipdfmx supports large Type1 files (>256 characters per font); I don't know the status of its present or planned support for the other two.

Some fonts (like standard Type1 fonts) contain only a standard palette of 256 character-slots. In general, such fonts do not contain expert characters or glyphs such as 'ff', 'ffi', and 'ffl'. Given a standard font, we need to combine information from at least one other corresponding font to get a complete and professional typeface for that standard font. There are three ways to prepare the raw fonts for installation. One may use:

1. the fontinst package (for Type1 `*.pfb's`);
2. FontForge (formerly PfaEdit) (for Type1 `*.pfb's`); and
3. pre-prepared fonts, with standard, expert, and variant glyphs all in one font (TrueType and, more and more, OpenType).

If your fonts are already in a pre-prepared format, then you may just skim the first two subsections below.

fontinst

ConTeXt has its own font installation script, texfont. From page 1 of the texfont manual (`mtexfont.pdf`):

> The script only covers 'normal' fonts... Special fonts, like expert fonts, assume a more in depth knowledge of font handling. We may deal with them in the future. The more demanding user can of course fall back on more complicated tools like fontinst.

Although written in PlainTeX, the interface to fontinst is somewhat LaTeX-oriented. So its syntax largely follows the NFSS. This is no problem for ConTeXt: we only need the virtual fonts and tfm's produced by fontinst, and we ignore the `*.fd` file. Below we outline the procedure for preparing the fonts for installation using fontinst.[3]

Assuming that you are starting with 256-character Type1 fonts, you may rename them according to the older Berry convention.[4] We don't need that convention with today's operating systems but we will use it as a starting point. This is since LaTeX already has a setup for Minion Pro that uses the Berry fontname scheme and some readers may already have the raw fonts in this format.

The Minion Pro that I have contains 31 fonts. Here is a descriptive listing of the Type1 Minion Pro family:

```
Minion (31 fonts):

pmnb7d.pfb      Minion Bold Oldstyle Figures
pmnb8a.pfb      Minion Bold
pmnb8x.pfb      Minion Bold Expert

pmnbi7d.pfb     Minion Bold Italic Oldstyle Figures
pmnbi8a.pfb     Minion Bold Italic
pmnbi8x.pfb     Minion Bold Italic Expert
```

pmnc7d.pfb	Minion Black Oldstyle Figures
pmnc8a.pfb	Minion Black
pmnc8x.pfb	Minion Black Expert
pmnr8a.pfb	Minion Regular
pmnr8x.pfb	Minion Regular Expert
pmnrc8a.pfb	Minion Regular Small Caps & Oldstyle Figures
pmnrd8a.pfb	Minion Regular Display
pmnrd8x.pfb	Minion Regular Display Expert
pmnrdc8a.pfb	Minion Regular Display Small Caps & Oldstyle Figures
pmnrdi8a.pfb	Minion Italic Display
pmnrdi8x.pfb	Minion Italic Display Expert
pmnrdic8a.pfb	Minion Italic Display Small Caps & Oldstyle Figures
pmnrdiw8a.pfb	Minion Italic Display Swash
pmnri8a.pfb	Minion Italic
pmnri8x.pfb	Minion Italic Expert
pmnric8a.pfb	Minion Italic Small Caps & Oldstyle Figures
pmnriw8a.pfb	Minion Italic Swash
pmnrp8a.pfb	Minion Ornaments
pmns8a.pfb	Minion Semibold
pmns8x.pfb	Minion Semibold Expert
pmnsc8a.pfb	Minion Semibold Small Caps & Oldstyle Figures
pmnsi8a.pfb	Minion Semibold Italic
pmnsi8x.pfb	Minion Semibold Italic Expert
pmnsic8a.pfb	Minion Semibold Italic Small Caps & Oldstyle Figures
pmnsiw8a.pfb	Minion Semibold Italic Swash

Let us begin our analysis of Minion; we need to make a few decisions. Just make a note of them for later; it helps to stay organized with all the accounting involved in the typescripts:
We first note that, aside from the ornamental font, there are 5 main style variations: medium, semibold, bold, black, and italic. Medium has a display version, italic has a display version, bold has an italic version, and semibold has an italic version, for a total of nine variations. We need to make some sense of this in terms of optical scaling. For our future typescript, we will initially group some of these as follows:

- For `\tf`, let's try medium for sizes $<$ 17.3pt, and medium display for sizes $\geqslant$ 17.3pt;
- For `\bf`, try bold for sizes $\geqslant$ 8pt, and black for sizes $\leqslant$ 8pt (there is no display size for bold). Similarly for `\bi`;
- For `\it`, we try italic for sizes $<$ 17.3pt, and italic display for sizes $\geqslant$ 17pt;[5]
- We will leave semibold as its own alternative, although I did once try treating semibold (`\sb`) as an option for small or caption-sizes ($\leqslant$ 8pt). I think this was a failure, but the reader should try it and judge for himself.

The rest of our choices will be analogous.

We also note the following, based on a direct examination of these fonts:

- Based on the above grouping, small caps will be available in both weights for `\tf` and for `\it`, but not for `\bf` (sigh) or `\bi`. Oddly, semibold has both a small caps variation and a small caps italic variation. According to Lehman (page 63), semibold is the actual default bold weight; maybe he's right.[6]
- For a given optical size (as tentatively defined above), old style figures are available in both the expert font and in the old style figures font;
- For some reason, Minion Bold Oldstyle Figures as well as Minion Bold Italic Oldstyle Figures have no small caps; each are identical to Minion Bold and Minion Bold Italic respectively, except for the numerals. The other styles use small caps in their old style figures versions;
- It is our intention to make old style numerals the default for our entire typeface collection; this makes sense in the humanities, I think;[7]
- For all five primary variations (`\tf`, `\it`, `\bf`, `\bi`, and `\sb`) and their derivatives, we will try
  — using the expert fonts for both old figures and expert ligatures;
  — using the old style figures fonts for small caps only (`\tf` and `\it`);
  Although we could just default to the old style figures fonts for `\bf` and `\bi`, for consistency purposes we will, for the time being, treat all four typefaces equal in this regard. You can always change this...
- The most difficult task to accomplish the above is dealing with the expert fonts in this collection. They share an non-standard encoding vector. We need to make our typeface collection default to the expert ligatures and to the old style numerals.

Preparing the raw fonts for installation involves making a fontinst file `makeminion.tex` like the following:

```
\input fontinst.sty

\installfamily{T1}{pmn}{}
\installfonts

% minionr
\installfont{minionr10}   {pmnr8a,pmnr8x,latin}     {T1j}{T1}{minion}{m}{n}{}
\installfont{minionr17}   {pmnrd8a,pmnrd8x,latin}   {T1j}{T1}{minion}{m}{n}{}

% minioni
\installfont{minioni10}   {pmnri8a,pmnri8x,latin}   {T1j}{T1}{minion}{m}{it}{}
\installfont{minioni17}   {pmnrdi8a,pmnrdi8x,latin} {T1j}{T1}{minion}{m}{it}{}

% minionb
\installfont{minionb10}   {pmnb8a,pmnb8x,latin}     {T1j}{T1}{minion}{b}{n}{}
\installfont{minionbl10}  {pmnc8a,pmnc8x,latin}     {T1j}{T1}{minion}{b}{n}{}

% minionbi
\installfont{minionbi10}  {pmnbi8a,pmnbi8x,latin}   {T1j}{T1}{minion}{m}{bi}{}

% minionsc
\installfont{minionsc10}  {pmnrc8a,latin}           {T1j}{T1}{minion}{m}{sc}{}
\installfont{minionsc17}  {pmnrdc8a,latin}          {T1j}{T1}{minion}{m}{sc}{}

% minionisci
\installfont{minionsci10} {pmnric8a,latin}          {T1j}{T1}{minion}{m}{sc}{}
\installfont{minionsci17} {pmnrdic8a,latin}         {T1j}{T1}{minion}{m}{sc}{}
```

```
% minionsb
\installfont{minionsb10}  {pmns8a,pmns8x,latin}     {T1j}{T1}{minion}{sb}{n}{}

% minionsbi
\installfont{minionsb10}  {pmnsi8a,pmnsi8x,latin}   {T1j}{T1}{minion}{sb}{it}{}

% minionsbsc
\installfont{minionsc8}   {pmnsc8a,latin}           {T1j}{T1}{minion}{sb}{sc}{}
\installfont{minionsci8}  {pmnsic8a,latin}          {T1j}{T1}{minion}{sb}{sc}{}

% minionisw
%\installfont{minionswi10}  {pmnriw8a,latin}          {T1j}{T1}{minion}{m}{it}{}
%\installfont{minionswi17}  {pmnrdiw8a,latin}         {T1j}{T1}{minion}{m}{it}{}
%\installfont{minionsbswi10}{pmnsiw8a,latin}          {T1j}{T1}{minion}{sb}{it}{}

% miniono
%\installfont{miniono10}   {pmnrp8a,latin}          {T1j}{T1}{minion}{m}{n}{}

\endinstallfonts

\bye
```

Let us look briefly at the first \installfont line (see Hoenig or Lehman for details):

- \installfont {minionr10}
  The name of our virtual font will be minionr10;
- {pmnr8a,pmnr8x,latin}
  Our standard font is pmnr8a, expert font is pmnr8x, and latin.mtx is the default fontinst metric file that defines at least 401 glyphs found in Latin alphabets (see Hoenig, page 180);
- {T1j}{T1}{minion}{m}{n}{}
  The *encoding file* is t1j.etx (Cork with oldstyle numerals), *general encoding* is T1 (cork), *family* is minion, *series* is medium, *shape* is normal, and size is left empty. This is all NFSS terminology.

Note that we have intentionally organized makeminion.tex to be analogous to our future typescript file. Also, we have commented out the swash and ornament lines. This is because I personally prefer to deal with the preparation, installation, and configuration of each of these two in its own directory, separate from the main fonts. So in the /swash subdirectory makeminionsw.tex will contain only the swash lines, and /ornaments will contain only the ornament line. Looking ahead, we will have three separate typescript classes: main, swash, and ornament. Because writing advanced typescripts requires a lot of careful accounting, it is better to keep these classes separate. If you don't believe me, try doing everything that follows in the configuration stage in a single typescript. You'll see;-)
In the LaTeX version, the final fonts are given names like pmnr9e instead of minionr10. Since we don't have to deal with NFSS and old encodings, we can happily dispense with that here.
Note: In retrospect, I prefer to avoid fontinst. There is a very limited number of pre-made *.etx files, though you can make your own. But if you have a set of 256-character-slots Type1 fonts, the next method will make our life a bit easier later, as you'll see. On the other hand, if you really need dvips, then you may need to go the fontinst route (dvipdfmx works fine with the next method).

FontForge

While I was messing with fontinst, the following thought occurred to me: is there some way we can merge the expert and standard fonts into a single font file, so we can just use texfont (you will soon see why this makes things easier)? I tried FontLab: no such feature. Fontographer? Foiled again. Then I looked at the open source FontForge (formerly PfaEdit).[8] For Windows users, there is a version for Cygwin. It's definitely worth installing a minimal Cygwin to have; instructions are on the FontForge site.

In FontForge, open `pmnr8a.pfb`. Then go to `ELEMENT => MERGE` to choose the corresponding expert font, `pmnr8x.pfb`. FontForge will add every character with a different name to the original glyph palette. Then save this new font to `minionr10.pfb`. You must repeat this for all standard fonts that have an expert companion. You can use `makeminion.tex` in the above subsection on fontinst to identify the correspondences and correct names. For those fonts that have no expert companion, just copy and rename them to our scheme.

A nice thing about FontForge is that it is scriptable. So those who are familiar with that can write a script so that FontForge can do all of this in batch. That skill is a bit beyond me, so I just did it the point-and-click way. Look up "scripting" in the FontForge documentation.

Pre-prepared Fonts

If you have OpenType or TrueType versions of the fonts then you are set. If you need to use aleph (ℵ, which cannot use `*.ttf/*.otf files`); or need dvipdfmx, then all you need to do is convert each font to Type1.[9] Don't worry about the 256-character-slot limit for Type1 fonts; it won't affect things for us. To follow along easily, save copies of your OpenType Minion Pro fonts to the names we are using here.

Installing the Fonts

First, we must have the `afm` files for all raw fonts. You can generate them with any decent font-editing software. There is an afm-generation utility, `getafm` that comes with TEXLive, but it does not procure the proper kerning info. A package of metrics for the *Adobe Type Classics for Learning* suite (including Minion Pro) is available from `http://www.lcdf.org/type/`.

fontinst

http://www.ntg.nl/maps/33/minion/fontinst.bat is a batch file that handles most of the work. Once you have generated the files and directories, you can install them in your local tree or in `/texmf-fonts`, which ConTEXt uses. You will also need a proper map file, which is where I made my big mistake with fontinst. For dvips I used lines like this:

```
pmnr8a     pmnr8a     <pmnr8a
```

But I discovered that I needed lines like this:

```
pmnr8a Minion-Regular "TeXBase1Encoding ReEncodeFont" <8r.enc<pmnr8a.pfb
```

Walter Schmidt has provided a complete LaTEX package for Minion:
`http://www.ctan.org/tex-archive/fonts/psfonts/w-a-schmidt/pmn.zip`
For details see Walter's package. Between this and Tutorial VI of Lehman's *Guide* you will learn all you need to know about installing Minion Pro, as well as a lot about fontinst. In any case, I much prefer using texfont. Our installation in texfont will involve multiple encodings. To do this in fontinst you may have to write your own `*.etx` files, endure a lot of debugging, and so forth. Make your life easy and get FontForge:-)

TEXfont: Type1, TrueType or OpenType Big Fonts

Let us begin by making three temporary directories:

□ `/main`
  Place all Minion `pfb`'s and `afm`'s here;
□ `/swash`
  Move all three swash fonts, `minionswi10.pfb`, `minionsbswi10.pfb`, and `minionswi17.pfb` here;
□ `/ornament`
  Move the ornament font `miniono10.pfb` and metric file here.

Here we set our encoding vectors for Minion Pro. We will use texnansi encoding as our base, though you can easily choose another (like ec) if you like. Actually, we will use the file `texnansi-lm.enc`, in /texmf-local/fonts/enc/dvips/lm, as our base file, because it is easier to edit than `texnansi.enc`. Just make sure to remove all `*.dup` extensions. For example, change `/OE.dup` to `/OE`.
Now we create a few encoding files (all go into `/main` except the last two). From careful study of these examples, you can easily make your own special encodings at will. **Note**: each encoding file must have precisely 256 character lines, not counting the beginning line and the ending line:

□ `texnansi-axo.enc`
  The prefix `texnansi` is important; it tells us that texnansi encoding is our foundation. The string 'ax' stands for 'Adobe Expert'. Finally, 'o' stands for 'old style numerals'. This encoding file will be used to create and install a virtual font that defaults to old style numerals. Simply replace the lines

```
/zero
/one
/two
/three
/four
/five
/six
/seven
/eight
/nine
```

  with

```
/zerooldstyle       %/zero etc.
/oneoldstyle        %:
/twooldstyle
/threeoldstyle
/fouroldstyle
/fiveoldstyle
/sixoldstyle
/sevenoldstyle
/eightoldstyle
/nineoldstyle
```

  The comments just remind us of the original characters we are replacing. Change the beginning line to `/enctexnansiaxo[`. There a couple of minor quirks to keep in mind. See http://www.ntg.nl/maps/33/minion/texnansi-axo.enc for the full `texnansi-axo.enc`; changes from the original `texnansi.enc` are noted. For example, there is no dottless 'j' in either the Adobe standard or expert encodings, at least not with Minion Pro.[10]

- □ texnansi-axu.enc
  This encoding file will be used to create and install a virtual font that defaults to upright numerals. Use the default numeral characters from texnansi.enc;
- □ texnansi-axs.enc
  This encoding file will be used to create and install a virtual font that defaults to superior numerals. Use /zerosuperior, etc.;
- □ texnansi-axi.enc
  This encoding file will be used to create and install a virtual font that defaults to inferior numerals. Use /zeroinferior, etc..
- □ texnansi-axuc.enc
  This encoding file will be used to create and install a virtual font that defaults to upright numerals and small caps. The small caps fonts that come with Minion Pro all default to old style numerals, and these numerals are encoded with the upright character names. Take texnansi-axu.enc and replace

  ```
  /a
  /b
  /c
  ```

  with

  ```
  /Asmall
  /Bsmall
  /Csmall
  ```

  and so forth. Using this particular encoding is only good for those standard fonts with small caps in the corresponding expert font. For example, Minion Bold Expert has no small caps (although Semibold Expert does). Basically you will be replacing all of the original small caps fonts with non-small caps big fonts encoded with small caps glyphs. We will say more about this below, in the section on typescripting.
- □ texnansi-ao.enc and texnansi-aw.enc
  We make encodings for the swashes and ornaments. The ornamnents take up 23 slots corresponding to A through W; the swashes take up A through Z. The /space slot is the only other one kept in place; fill up the rest with /.notdef's, e.g.,

  ```
  /.notdef
  /.notdef
  /.notdef
  /ornament1      %/A,
  /ornament2      %/B
  /ornament3      %/C, etc
  ```

  for ornaments, and

  ```
  /.notdef
  /.notdef
  /.notdef
  /A
  /B
  /C
  /D              % etc.
  ```

  for swashes.

There are lots of other possibilities, like an encoding that uses text-fractions and so forth. You are now in control!
It is now time to install. texfont will do most of the work, but you have to install the encoding files by hand. It would be nice if texfont could do this for us as well. In the meantime, copy the encoding files to `/texmf-fonts/fonts/enc/dvips/minion`.
Do Not forget to install the encoding files!
Now we are ready to install our main fonts with texfont. The directory `/main` should have the `*.pfb` files, the `*.afm` files, and the encoding files (or you can pre-install the encoding files and do TEXHASH). From each of the three respective directories, issue the corresponding commands from the following:

```
texfont --ma --in --en=texnansi-axo --ve=adobe --co=minion --show

texfont --ma --in --en=texnansi-axu --ve=adobe --co=minion --show

texfont --ma --in --en=texnansi-axs --ve=adobe --co=minion --show

texfont --ma --in --en=texnansi-axi --ve=adobe --co=minion --show

% texfont --ma --in --en=texnansi-axuc --ve=adobe --co=minion --show

% uncomment if small~caps with upright numerals are desired
```

Do `texfont --help` to see the meaning of each of the above switches. The above commands use abbreviated versions (first two letters) of these options.[11]
Now you will find four `pdf` files in `/main`: `texnansi-axo-adobe-minion.pdf`, `texnansi-axu-adobe-minion.pdf`, `texnansi-axs-adobe-minion.pdf`, and `texnansi-axi-adobe-minion.pdf`. Take a look at these; they include beautiful font charts of your encodings. Also take a look at the map files in `/texmf-fonts/fonts/map/pdftex/context`. Peruse especially the way the virtual fonts and `tfm` files fonts are named. It's verbose but very easy to read and systematic.
Did you remember to install the encoding files?

Configuration

The texnansi-axo Typface Collection

Now we need to generate a set of typescripts that can handle our main Minion font collection: let's call them `type-mino.tex`, `type-minu.tex`, `type-mins.tex`, and `type-mini.tex`. All four are almost identical so we will analyze one of them in detail, `type-mino`. Each typescript will have five main parts: *font mapping*, *general names*, *font sizes*, *map loading*, and *final typefaces*. Let us deal with each of these in turn.

- Font Mapping
  Here we map the raw fonts to easy-to-understand names. Note that we are mapping, not directly to the `pfb`'s, but to the virtual fonts.

```
% We need a few switches: I don't guarantee that
% they don't conflict with other commands;-)

\definestyle [italicsmallcaps,smallcapsitalic] [\si] []
\definestyle [black]                           [\bk] []
\definestyle [semiboldroman,semibold]          [\sb] []
\definestyle [semibolditalic]                  [\st] []
\definestyle [semiboldsmallcaps]               [\sp] []
\definestyle [semiboldsmallcapsitalic]         [\stp][]

% Regular serifs, greater than 8pt, less than 17.3pt
```

```
\starttypescript[serif]                    [miniono] [texnansi-axo]

\definefontsynonym [Minion10]              [texnansi-axo-minionr10]
\definefontsynonym [Minion17]              [texnansi-axo-minionr17]

\definefontsynonym [MinionItalic10]        [texnansi-axo-minioni10]
\definefontsynonym [MinionItalic17]        [texnansi-axo-minioni17]

\definefontsynonym [MinionBold10]          [texnansi-axo-minionb10]
\definefontsynonym [MinionBlack]           [texnansi-axo-minionbl10]

\definefontsynonym [MinionBoldItalic]      [texnansi-axo-minionbi10]

\definefontsynonym [MinionCaps10]          [texnansi-axu-minionsc10]
\definefontsynonym [MinionCaps17]          [texnansi-axu-minionsc17]

\definefontsynonym [MinionItalicCaps10]    [texnansi-axu-minionsci10]
\definefontsynonym [MinionItalicCaps17]    [texnansi-axu-minionsci17]

\definefontsynonym [MinionSemiBold]        [texnansi-axo-minionsb10]

\definefontsynonym [MinionSemiBoldItalic]  [texnansi-axo-minionsbi10]

\definefontsynonym [MinionSemiBoldCaps]    [texnansi-axu-minionsbsc10]

\definefontsynonym [MinionSemiBoldItalicCaps][texnansi-axu-minionsbsci10]

\stoptypescript
```

Note that we map to the upright-encoded fonts for the six fonts with small caps. This is because the small caps fonts each defaults to old style numerals, but those numerals are encoded in the font with upright names. Furthermore, the small caps fonts do not have corresponding experts. So the small caps virtual fonts in `texnansi-axo` encoding have no numerals at all. On the other hand, the `texnansi-axu` encoded small caps virtual fonts will display old style numerals because those numerals are encoded in the font with upright names. The rest will display upright numerals. This inconsistency is wholly due to the manufacturer of the original raw fonts.
Similarly, the typescript for `texnansi-axu` encoded fonts will need to map small caps to the `texnansi-axuc` encoded fonts, if full consistency is desired. The `texnansi-axuc` encoded fonts do not need their own typescript, since they are just meant to supplement texnansi-axu.[12]
Note the option `[miniono]`. For `type-minu.tex` it should be `[minionu]` (with a 'u') and so forth.

□ General Names
This part may seem redundant right now, but it will make sense when we add the Myriad Pro collection. That is a sans serif, while Minion is a serif, so this helps keep things clear and organized.

```
\starttypescript[serif]                    [miniono] [name]

\definefontsynonym [Serif]                 [Minion10]
\definefontsynonym [Serif17]               [Minion17]

\definefontsynonym [SerifItalic10]         [MinionItalic10]
\definefontsynonym [SerifItalic17]         [MinionItalic17]

\definefontsynonym [SerifBold10]           [MinionBold10]
\definefontsynonym [SerifBlack]            [MinionBlack]

\definefontsynonym [SerifBoldItalic]       [MinionBoldItalic]
```

```
\definefontsynonym [SerifCaps10]            [MinionCaps10]
\definefontsynonym [SerifCaps17]            [MinionCaps17]

\definefontsynonym [SerifItalicCaps10]      [MinionItalicCaps10]
\definefontsynonym [SerifItalicCaps17]      [MinionItalicCaps17]

\definefontsynonym [SerifSemiBold]          [MinionSemiBold]

\definefontsynonym [SerifSemiBoldItalic]    [MinionSemiBoldItalic]

\definefontsynonym [SerifSemiBoldCaps]      [MinionSemiBoldCaps]

\definefontsynonym [SerifSemiBoldItalicCaps][MinionSemiBoldItalicCaps]

\stoptypescript
```

□ Font Sizes
This is where we implement optical scaling (what little there is, anyway). If you have Minion Pro Opticals, you will have more choices. The following typescript will give you the needed insight to implement your own scheme for optical scaling.
Note that in the first line of this section of our typescript, we have mapped `Minion10` to, not `Serif10`, but to just `Serif`. ConTeXt treats the `Serif` font as the default or empty font; if it is not defined, in a few cases ConTeXt will fall back to a typeface where it is defined (generally **Latin Modern**).[13]

```
\starttypescript [serif] [miniono] [size]

\definebodyfont [9pt,10pt,11pt,12pt,14.4pt]
  [rm]
  [tf=Serif sa 1,
   sc=SerifCaps10 sa 1,
   it=SerifItalic10 sa 1,
   si=SerifItalicCaps10 sa 1]

\definebodyfont [4pt,5pt,6pt,7pt,8pt]
  [rm]
  [tf=Serif sa 1,
   sc=SerifCaps10 sa 1,
   it=SerifItalic10 sa 1,
   si=SerifItalicCaps10 sa 1,
   bf=SerifBlack sa 1]

\definebodyfont [17.3pt,20.7pt,24.9pt]
  [rm]
  [tf=Serif17 sa 1,
   sc=SerifCaps17 sa 1,
   it=SerifItalic17 sa 1,
   si=SerifItalicCaps17 sa 1]

\definebodyfont [9pt,10pt,11pt,12pt,14.4pt,17.3pt,20.7pt,24.9pt]
  [rm]
  [bf=SerifBold10 sa 1]

\definebodyfont
  [24.9pt,20.7pt,17.3pt,14.4pt,12pt,11pt,10pt,9pt,8pt,7pt,6pt,5pt,4pt]
  [rm]
  [bi=SerifBoldItalic sa 1,
   sb=SerifSemiBold sa 1,
   st=SerifSemiBoldItalic sa 1,
   sp=SerifSemiBoldCaps sa 1,
   stp=SerifSemiBoldItalicCaps sa 1]

\stoptypescript
```

Note: Some of these switches are newly defined (like `\sp`), and the ConTEXt mechanism for enlarging and reducing the size of a given style variation will not work. We need to define them for completeness. See pages 129–131 of *ConTEXt: the Manual* for details. It's really quite straightforward, just a bit tedious and verbose, so we leave it as an exercise for the reader.[14] See also the typescript in http://www.ntg.nl/maps/33/minion/type-mpoo.tex. Choosing optical sizes is an area that needs a bit of experimenting to get exactly right. For example, does the black font really work at small bold sizes?

- Map Loading
  Here we load our map files, created during installation.

```
\starttypescript[map] [miniono] [texnansi-axo]

\loadmapfile[texnansi-axo-adobe-minion.map]
\loadmapfile[texnansi-axu-adobe-minion.map]

\stoptypescript
```

Here we also need the `texnansi-axu` map for the small caps as discussed above. Note that `type-minu.tex` will also need to load the `texnansi-axuc` map file, if you have installed and desire to have small caps with upright numerals.[15]

- Final Typefaces
  This is where we put it all together, our Minion typeface collection. We also define those fonts that do not come with Minion, Myriad, or Poetica, such as math fonts (we use Euler) and monospaced (we use Latin Modern). Note the 'o' suffix in what follows. Such identifying suffixes will be needed in the other typescript files as well as well.
  Note: While very powerful and transparent, typescripts are quite sensitive to these kinds of seemingly minor accounting issues, so be careful.

```
\starttypescript[ADOBEMiniono]

\definebodyfontenvironment
  [adobeminiono]
  [default]
  [interlinespace=2.6ex]

\definetypeface [adobeminiono]
[rm] [serif] [miniono] [miniono] [encoding=texnansi-axo]

% Myriad and Poetica to be configured later, then uncomment
%\definetypeface [adobeminiono]
%[ss] [sans] [myriado] [myriado] [encoding=texnansi]

%\definetypeface [adobeminiono]
%[cg] [calligraphy] [poetica] [poetica] [encoding=texnansi]

\definetypeface [adobeminiono]
[mm] [math] [euler] [default] [encoding=texnansi,rscale=0.89]

\definetypeface [adobeminiono]
[tt] [mono] [modern] [default] [encoding=texnansi,rscale=0.99]

\stoptypescript
```

We note that the non-Minion fonts used in our typeface collection, such as Euler math fonts and Latin Modern monospaced, need to be scaled. That is what the `rscale=<scale factor>` option does for us. We also note that Minion needs a smaller interline space factor than the usual 2.8`ex`. We may need to do some more testing in this regard, though 2.6`ex` seems to work well for the Minion design.

Note: Be aware that ConTeXt sets up `\em` with the slanted (`\sl`) style variation by default. But Minion Pro does not come with a slanted font. So `\em` will not work unless you map one of your fonts – see the previous section on size definitions – to `\sl`. Declare

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

either in your typescript or in your style/environment file. It may not be such a good idea to define it in the typescript, because you could get odd results depending on the order your typescripts are scanned during compilation (assuming you've setup `\em` differently somewhere else).

Now write the above set of typescripts to a file, `type-mino.tex`. We can now test our typescript so far. Here is a test file:

```
% output=pdf interface=en

\usetypescriptfile[type-mino]
\usetypescript[ADOBEMiniono]
\switchtotypeface[adobeminiono]%

\starttext

This is a test of Minion in \CONTEXT. 1234

\bf This is a test of Minion in \CONTEXT. 1234

\it This is a test of Minion in \CONTEXT. 1234

\bi This is a test of Minion in \CONTEXT. 1234

\sc This is a test of Minion in \CONTEXT. 1234

\si This is a test of Minion in \CONTEXT. 1234

\sb This is a test of Minion in \CONTEXT. 1234

\stp This is a test of Minion in \CONTEXT. 1234

\switchtotypeface[adobeminionor]
{\tf ABCDEFGHIJKLMNOPQRSTUVW \par}\blank

\switchtotypeface[adobeminionsw]
{\sw ABCDEFGHIJKLMNOPQRSTUVW \par}\blank

\stoptext
```

Compiling gives us:

This is a test of Minion in ConTeXt. 1234

This is a test of Minion in ConTeXt. **1234**

This is a test of Minion in ConTeXt. *1234*

This is a test of Minion in ConTeXt. ***1234***

THIS IS A TEST OF MINION IN ConTeXt. 1234

THIS IS A TEST OF MINION IN ConTeXt. *1234*

This is a test of Minion in ConTeXt. **1234**

THIS IS A TEST OF MINION IN ConTeXt. ***1234***

ABCDEFGHIJKLMNOPQRSTUVW

Sample application

We end with an application: In our Minion installation and configuration we have a superior numerals typeface. This looks better than either upright or old style numerals for footnote marking. This example compares superior and upright numerals in footnotes, with old style numerals in the running text:

Consider what is said, not who has said it.[1234]

[1234] This aphorism is by ^{c}Alī ibn Abī Ṭālib (d. 661CE/AH).

Intermezzo 1 Upright footnote numerals.

Consider what is said, not who has said it.[1234]

[1234] This aphorism is by ^{c}Alī ibn Abī Ṭālib (d. 661CE/AH).

Intermezzo 2 Superior footnote numerals.

Post-dvi processing

Unfortunately, inconsistencies between pdfe-TeX, dvips, and dvipdfmx mean we have to do more work if we need post-`dvi` processing for any reason (this is the case with ℵ, for example).

dvipdfmx

You will need to

- □ write at least one map file for your collection;
- □ Look at, e.g., `texnansi-axo-adobe-minion.map`. Change the syntax from

```
texnansi-axo-raw-minionr10 Minion-Regular 4 < minionr10.pfb texnansi-axo.enc
```

to

```
texnansi-axo-raw-minionr10 texnansi-axo minionr10
```

- Now make your map file available to dvipdfmx. You can add a line like

  ```
  f minion-dvipdf.map
  ```

  to the file `/texmf-local/fonts/dvipdfm/config/config`, or you may call your map file from the command line

  ```
  $ dvipdfmx -f minion-dvipdf.map
  ```

dvips

If you need to use dvips, you may have to go the fontinst route. This is because dvips apparently looks for Type1 fonts with a 256-glyph limit, and ours (as well as **Latin Modern**) are bigger. This limitation seems a bit outdated, and hopefully the maintainers of dvips will one day remove this limitation. In any case, see Tutorial VII of Lehman's *Guide* for a very thorough discussion of preparing map files for dvips with fontinst.

I hope that you have found this article clear and enjoyable. See also *This Way* # 9, Using Platform Fonts, by Hans Hagen, for more on font installation.[16]
Best wishes for painless font installation in ConTEXt![17]

Minion Pro Opticals

As I was finishing this issue I received the complete Minion Pro Opticals set, in OpenType format. This set is more internally coherent than the older version we used for this tutorial. It contains six style variations: medium or regular, semibold, bold, italic, semibold italic, and bold italic. The black style variation is apparently gone. Each style variation comes in four optical sizes: normal, caption, subhead, and display. Each font has a standard 256-character encoding, plus a set of old style numerals, superiors, inferiors, a set of small caps (we have bold small caps now!), ornaments and hundreds of other alternates. There is no dedicated small caps or ornaments font. Each italic font has a large palette of swashes, many more than the original swash fonts. There is also Greek, Cyrillic, and lots of alternate or fancy ligatures. This is really much better than the original set.
Our encodings prepared earlier will suffice with a few changes (and you can always make your own):

- `texnansi-axo.enc`, `texnansi-axu.enc`, `texnansi-axs.enc`, and `texnansi-axi.enc` will stay the same;
- For small caps we now need a `texnansi-axoc.enc` for small caps with old style numerals. Just modify `texnansi-axuc.enc` and replace the default numerals with the old style ones;
- `texnansi-aw.enc` will have to change `/Aswash` to `/A.swash`, etc.. The italics font also offer lots of swash capitals with accents. Some of these swashes do not have a corresponding entry in the standard encoding: for example, there is not `/Ebreve` in the standard encoding to match `/Ebreve.swash`. So if you want the esoteric swashes you will have to pick and choose how you want to encode this within a 256-character context.

One idea about swashes: since they are now part of the full italics fonts, treat them like small caps, and encode them in the same `/a--/z` band of the encoding. This was much less trivial to accomplish in the old fonts.

- □ The names of the ornaments in `texnansi-ao.enc` will have to be changed: `/ornament1` becomes `/orn.001`, etc., up to `/orn.023`. An identical set of ornaments is present in every font, so you can also encode ornaments like a small caps font if you like.

Installation is just as before. Convert fonts to `*.pfb` (with `*.afm`), place in a separate directory with your encodings, then run texfont for as many encodings as you like.[18]

The typescript files are mostly as before: the only really interesting difference is the much better optical scaling. According to the Minion Pro Opticals documentation, the intended optical scaling spectrum is as follows:

- □ Caption: 6–8.4 point
- □ Normal (Regular): 8.5–13 point
- □ Subhead: 13.1–19.9 point
- □ Display: 20+ point

For small caps, one may choose to write a separate typescript file and typeface collection, in which case one has to switch fonts to use small caps. Or one can integrate, e.g, the small caps fonts into the upright numerals typescript file. Experiment to get the combination that works best for you.

We give the typescript files different names from before: `type-mpoo` for Minion Pro Opticals old style, `type-mpou` for upright, and so forth.

One full possible typescript, that for `type-mpoo`, including small caps, is available from http://www.ntg.nl/maps/33/minion/type-mpoo.tex. It should reward careful study.

Enjoy!

Notes

1. For details, see “Designing Multiple Master Typefaces,” by Adobe: `http://partners.adobe.com/public/developer/en/font /5091.Design_MM_Fonts.pdf`.

2. We use the expressions ‘style’, ‘variation’, and ‘family’ in the senses employed in *ConTeXt: the Manual*, page 91. Adobe Minion Pro is a font *family* or typeface *family*, roman and sans serif are *styles*, bold and italic are *style variations*. In the ConTeXt world, the expression ‘typeface’ is often used to mention a user-defined collection of fonts, often drawn from various families.

3. For a wealth of details about fontinst and virtual fonts, see Alan Hoenig’s book *TeX Unbound*. A more recent and up-to-date manual is *The Font Installation Guide*, by Phillip Lehman. It is available in CTAN:/info/type1fonts/fontinstallationguide.

4. For details, see Hoenig, pages 132–134, and Lehman, pages 11–13.

5. This is all intentionally experimental. Lehman, page 63 has more professional suggestions, but I think it’s important to reflect ourselves. Probably you will one day have to install a font where no one has made predeterminations about this sort of thing.

6. On the other hand, Minion Pro Opticals has small caps for bold, and the official documentation seems to indicate that the default bold is, indeed, Minion Bold.

7. In *The Elements of Typographic Style*, Bringhurst enjoins:

> *Use titling [upright] figures with full caps, and text [old style] figures in all other circumstances.*

8. Available here: `http://fontforge.sourceforge.net/` .

9. One may use FontForge for this. There is also a tool `cfftot1` provided by lcdf (`http://www.lcdf.org/type/`) but it can not, as far as I can tell, generate an `*.afm` file. But see Appendix 1.

10. There is a free tool, `t1dotlessj`, that creates a dotless-‘j’ Type1 font from an existing standard font:

`http://www.lcdf.org/type/t1dotlessj.1.html` .
You may then use FontForge to merge this with your main font (preferable), or go through fontinst.
11. Adam Lindsay pointed out to me that you may also use the `--variant` option (e.g., `--va=texnansi-axo` instead of `--en`. Then there may be no need to install the encoding files: just use `[encoding=texnansi]` in the typescripts.
12. In `type-minu.tex`, replace the raw font names in the following lines

```
\definefontsynonym [MinionCaps10]            [texnansi-axu-minionsc10]
\definefontsynonym [MinionCaps17]            [texnansi-axu-minionsc17]

\definefontsynonym [MinionItalicCaps10]      [texnansi-axu-minionsci10]
\definefontsynonym [MinionItalicCaps17]      [texnansi-axu-minionsci17]

\definefontsynonym [MinionSemiBoldCaps]      [texnansi-axu-minionsbsc10]

\definefontsynonym [MinionSemiBoldItalicCaps] [texnansi-axu-minionsbsci10]
```

with the corresponding names from the `texnansi-axuc`:

```
\definefontsynonym [MinionCaps10]            [texnansi-axuc-minionr10]
\definefontsynonym [MinionCaps17]            [texnansi-axuc-minionr17]

\definefontsynonym [MinionItalicCaps10]      [texnansi-axuc-minioni10]
\definefontsynonym [MinionItalicCaps17]      [texnansi-axuc-minioni17]

\definefontsynonym [MinionSemiBoldCaps]      [texnansi-axuc-minionsb10]

\definefontsynonym [MinionSemiBoldItalicCaps] [texnansi-axuc-minionsbi10]
```

13. My thanks to Adam Lindsay for pointing this out.
14. Here is one example to get you started:

```
\definebodyfont
[24.9pt,20.7pt,17.3pt,14.4pt,12pt,11pt,10pt,9pt,8pt,7pt,6pt,5pt,4pt]
[rm]
[sp=SerifSemiBoldCaps sa 1,
 spa=SerifSemiBoldCaps scaled \magstep1, % or sa a
 spb=SerifSemiBoldCaps scaled \magstep2, % or sa b
 spc=SerifSemiBoldCaps scaled \magstep3, % or sa c
 spd=SerifSemiBoldCaps scaled \magstep4] % or sa d
```

and so forth.
15. That is, you will need to declare something like

```
\starttypescript[map] [minionu] [texnansi-axu]

\loadmapfile[texnansi-axu-adobe-minion.map]
\loadmapfile[texnansi-axuc-adobe-minion.map]

\stoptypescript
```

in `type-minu.tex`.
16. `http://pragma-ade.com/general/magazines/mag-0009.pdf`
17. I would like to especially thank Hans Hagen, Adam Lindsay, Thomas A.Schmitz, Ralf Stubner, and others from the ConTEXt mailing list for their help and assistance during the struggle to prepare this issue.
18. For an alternative approach, see Adam Lindsay's "OpenType installation basics for ConTEXt" in *The PracTEX Journal* 2005 No 02: `http://tug.org/pracjourn/`. It makes use of the `cfftot1` utility mentioned in footnote 9.

Idris Samawi Hamid

Hyphenation Patterns

Pattern files

TEX has two mysterious commands that the average user will never or seldom meet:

```
\hyphenation{as-so-ciates}
\patterns   {.ach4}
```

Both commands can take multiple strings, so in fact both commands should be plural. The first command can be given any time and can be used to tell TEX that a word should be hyphenated in a certain way. The second command can only be issued when TEX is in virgin mode, i.e. starting with a clean slate. Normally this only happens when a format is generated.

The second command is more mysterious than the first one and its entries are a compact way to tell TEX between what character sequences it may hyphenate words. The numbers represent weights and the (often long) lists of such entries are generated with a special program called `patgen`. Since making patterns is work for specialists, we will not go into the nasty details here.

In the early stage of ConTEXt development it came with its own pattern files. Their names started with `lang-` and their suffixes were `pat` and `hyp`.

However, when ConTEXt went public, I was convinced to drop those files and use the files already available in distributions. This was achieved by using the ConTEXt filename remapping mechanism. Although those files are supposed to be generic, this is not always the case, and it remains a gamble if they work with ConTEXt. Even worse, their names are not consistent and the names of some files as well as locations in the tree keep changing. The price ConTEXt users pay for this is lack of hyphenation until such changes are noticed and taken care of. Because constructing the files is an uncoordinated effort, all pattern files have their own characteristics, most noticably their encoding.

After the need to adapt the name mapping once again, I decided to get back to providing ConTEXt specific pattern files. Pattern cooking is a special craft and TEX users may count themselves lucky that it's taken care of. So, let's start with thanking all those TEX experts who dedicate their time and effort to get their language hyphenated. It's their work we will build (and keep building) upon.

In the process of specific ConTEXt support, we will take care of:

- consistent naming, i.e. using language codes when possible as a prelude to a more sophisticated naming scheme, taking versions into account
- consistent splitting of patterns and hyphenation exceptions in files that can be recognized by their suffix
- making the files encoding independent using named glyphs
- providing a way to use those patterns in plain TEX as well

Instead of using a control sequence for the named glyphs, we use a different notation:

```
[ssharp] [zcaron] [idiaeresis]
```

The advantage of this notation is that we don't have to mess with spacing so that parsing and cleanup with scripts becomes more robust. The names conform to the ConTEXt way of naming glyphs and the names and reverse mappings are taken from the encoding files in the ConTEXt distribution, so you need to have ConTEXt installed.

The ConTEXt pattern files are generated by a ruby script. Although the converting is rather straightforward, some languages need special treatment, but a script is easily adapted. If you want a whole bunch of pattern files, just say:

```
ctxtools --patterns all
```

or, if you want one language:

```
ctxtools --patterns nl
```

If for some reason this program does not start, try:

```
texmfstart ctxtools --patterns nl
```

When things run well, this will give you four files:

lang-nl.pat	the patterns in an encoding indepent format
lang-nl.hyp	the hyphenation exceptions
lang-nl.log	the conversion log (can be deleted afterwards)
lang-nl.rme	the preambles of the files used (copyright notices and such)

If you redistribute the files, it makes sense to bundle the `rme` files as well, unless the originals are already in the distribution. It makes no sense to keep the log files on your system. When the file `lang-all.xml` is present, the info from that file will be used and added to the pattern and hyphenation files. In that case no `rme` and `log` file will be generated, unless `--log` is provided.

In the Dutch pattern file you will notice entries like the following:

```
e[ediaeresis]n3
```

So, instead of those funny (encoding specific) `^^fc` or (format specific) `\"e` we use names. Although this looks ConTEXt dependent it is rather easy to map those names back to characters, especially when one takes into account that most languages only have a few of those special characters and we only have to deal with lower case instances.

The ConTEXt support module `supp-pat.tex` is quite generic and contains only a few lines of code. Actually, most of the code is dedicated to the simple XML handler. Loading a pattern meant for EC encoded fonts in another system than ConTEXt is done as follows:

```
\bgroup

\input supp-pat

\lccode"E4="E4 \definepatterntoken adiaeresis ^^e4
\lccode"F6="F6 \definepatterntoken odiaeresis ^^f6
\lccode"FC="FC \definepatterntoken ediaeresis ^^fc
\lccode"FF="FF \definepatterntoken ssharp     ^^ff

\enablepatterntokens
\enablepatternxml

\input lang-de.pat
\input lang-de.hyp

\egroup
```

In addition to this one may want to set additional lower and uppercase codes. In ε-TEX these are stored with the language.

Just for completeness we provide the magic command to generate the XML variants:

```
ctxtools --patterns --xml all
```

This will give you files like:

```
<?xml version='1.0' standalone='yes'?>

<!-- some comment -->

<patterns>
... e&ediaeresis;n3 ...
</patterns>
```

This is also accepted as input but for our purpose it's probably best to stick to the normal method. The pattern language is a TEX specific one anyway.

Installing languages

Installing a language in ConTEXt should not take too much effort assuming that the language is supported. Language specific labels are grouped in `lang-*` files, like `lang-ger.tex` for the germanic languages.

Patterns will be loaded from the files in the general TEX distribution unless `lang-nl.pat` is found, in which case ConTEXt assumes that you prefer the ConTEXt patterns. In that case, run

```
ctxtools --patterns all
```

You need to move the files to the ConTEXt base path that you can locate with:

```
textools --find context.tex
```

You can also use `kpsewhich`, but the above method does an extensive search. Of course you can also generate the files on a temporary location. Now it's time to generate the formats:

```
texexec --make --all
```

Since XeTEX needs patterns in utf-8 encoding, we provide a switch for achieving that:

```
texexec --make --all --utf8
```

Beware: you need to load patterns for each language and encoding combination you are going to use. You can configure your local `cont-usr` file to take care of this. When an encoding does not have the characters that are needed, you will get an error. When using the non ConTEXt versions of the patterns this may go unnoticed because the encoding is hard coded in the file. Of course it will eventually get noticed when the hyphenations come out wrong.

The ConTEXt distribution has a file lang-all.xml that holds the copyright and other notes of the patterns. A description looks like:

```
<description language='nl'>
  <sourcefile>nehyph96.tex</sourcefile>
  <title>TeX hyphenation patterns for the Dutch language</title>
  <copyright>
    <year>1996</year>
    <owner> Piet Tutelaers (P.T.H.Tutelaers@tue.nl)</owner>
    <comment>8-bit hyphenation patterns for TeX based upon the
      new Dutch spelling, officially since 1 August 1996.
      These patterns follow the new hyphenation rules in the
      'Woordenlijst Nederlandse Taal, SDU Uitgevers, Den Haag
      1995' (the so called 'Groene Boekje') described in
      section 5.2 (Het afbreekteken)</comment>
  </copyright>
</description>
```

This file is 'work in process': more details will be added and comments will be enriched.

Commands

You can at any moment add additional hyphenation exceptions to the language specific dictionaries. For instance:

```
\language[nl] \hyphenation{pa-tiën-ten}
```

Switching to another language is done with the `\language` command. The document language is set with `\mainlanguage`.

If you want to let TEX know that a word should be hyphenated in a special way, you use the `\-` command, for instance:

```
Con\-TeXt
```

Compound words are not recognized by the hyphenation engine, so there you need to add directives, like:

```
the ConTeXt|-|system
```

If you are using XML as input format, you need to load the hyphenation filter module. Here we assume that utf encoding is used:

```
\useXMLfilter[utf,hyp]
```

In your XML file you can now add:

```
<hyphenations language='nl' regime='utf'>
  <hyphenation>pa-tiën-ten</hyphenation>
  <hyphenation>pa-tiën-ten-or-ga-ni-sa-tie</hyphenation>
  <hyphenation>pa-tiën-ten-plat-form</hyphenation>
</hyphenations>
```

This filter also defines some auxiliary elements. Explicit hyphenation points can be inserted as follows:

```
Zullen we hier af<hyphenate/>bre<hyphenate/>ken of niet?
```

The compound token can be anything, but keep in mind that some tokens are treated special (see other manuals).

```
Wat is eigenlijk een patiënten<compound token="-"/>platform?
```

A language is set with:

```
nederlands <language code="en">english</language> nederlands
```

If you set attribute `scope` to `global`, labels (as used for figure captions and such) adapt to the language switch. This option actually invokes `\mainlanguage`.

Languages

When users in a specific language area use more than one font encoding, patterns need to be loaded multiple times. In theory this means that one can end up with more instances than TEX can host. However, the number of sensible font encodings is limited as is the number of languages that need hyphenation. Now that memory is cheap and machines are fast, preloading a lot of pattern files is no problem. The following table shows the patterns that are preloaded in the version of ConTEXt that is used to process this file.

language	encoding	mapping	number	left$_{\text{min}}$	right$_{\text{min}}$
nl	texnansi	texnansi	1	2	2
nl	ec	ec	2	2	2
fr	texnansi	texnansi	3	2	2
fr	ec	ec	4	2	2
de	texnansi	texnansi	5	2	2
de	ec	ec	6	2	2
it	texnansi	texnansi	7	2	2
it	ec	ec	8	2	2
pt	texnansi	texnansi	9	2	2
pt	ec	ec	10	2	2
hr	ec	ec	11	2	2
pl	pl0	pl0	12	2	2
pl	ec	ec	13	2	2
pl	qx	qx	14	2	2
cz	il2	il2	15	2	2
cz	ec	ec	16	2	2
sk	il2	il2	17	2	2
sk	ec	ec	18	2	2
sl	il2	il2	19	2	2
sl	ec	ec	20	2	2
en	ec	ec	22	2	2
da	ec	ec	23	2	2
sv	ec	ec	24	2	2
af	ec	ec	25	2	2
no	ec	ec	26	2	2
deo	ec	ec	27	2	2
uk	ec	ec	28	2	2
us	ec	ec	29	2	2
es	ec	ec	30	2	2
ca	ec	ec	31	2	2
la	ec	ec	32	2	2
ro	ec	ec	33	2	2
tr	ec	ec	34	2	2
fi	ec	ec	36	2	2
hu	ec	ec	37	2	2

In the (near) future the somewhat arcane `pl0` *and* `il2` *encodings will go away since they are only used for Polish and Czech/Slovak computer modern fonts, which can be replaced by Latin Modern alternatives. Also, a new dense encoding may find its way into this list.*

Hans Hagen

What do you do with ConTEXt?

Abstract
User responses to the question: "What do you do with ConTEXt?".

This message was posted (by me) on the ntg-context mailing list on November 4, 2005.

> Hans and I believe it would be nice to publish the collected responses to this simple question:
>
> What do you do with ConTEXt?
>
> We think it would be nice to see all the various ways in which people experience ConTEXt. We are not looking for articles, in fact we really want just a small amount of text per user, nothing longer then you would normally type in an email message.

The following article is a collection of the replies I received. I have not performed editorial corrections except formatting, but the majority of the replies has been shortened for reasons of length or pointedness. What is left is an overview of the kind of typesetting projects people use ConTEXt for. I hope you find the results interesting.

Taco Hoekwater (editor)

Nikolai Weibull
I use ConTEXt for any document that I figure someone will want to print out on paper some day. This includes resumes, letters, and articles. I even considered typesetting software documentation with ConTEXt, but haven't yet decided if PDFs are the right medium for that kind of text. I also typeset my master's thesis using ConTEXt and probably spent as much time hacking ConTEXt as I did on doing "actual work" for the content. Not because ConTEXt couldn't do what I wanted, but because ConTEXt allowed me to do anything I wanted.

Mari Voipio
I first learned ConTEXt because of work:
My employer makes measuring instruments for industry and I write and/or compile and layout the end-user manuals for those instruments. Traditionally, those manuals were written and edited in Word, but when the mass of text and the amount of figures grew, it was obvious that I needed a new tool before becoming a total wreck due to corrupted and unruly Word files. A workmate had written his thesis in TEX and started to look for various TEX based solutions, of which I finally chose ConTEXt.

Duncan Hothersall
I have a repository of learning materials which are marked up in (DocBook-based) XML. They are in a few different languages (majority English, with Spanish, Dutch and recently Chinese) and belong to several different universities/training organisations on whose behalf we maintain them. I convert the XML into ConTEXt code in a batch process using a text processing language, and then use ConTEXt to typeset the result into books and booklets of different styles. We also produce web versions of a lot of the materials, and I use ConTEXt (plus ImageMagick) to create equation GIFs for maths that can't be rendered in pure HTML.

Luigi Scarso
1. price-lists from XML (DB exports) ;
2. labels for shoes and dresses, from XML/CSV;
3. data sheets (barcodes, code number and so on), from XML/CSV;

David Wooten
I'm using ConTEXt to typeset my Ph.D. dissertation, and have used it over the last few years to typeset seminar papers, outlines, and so on. Recently I've also helped a friend typeset his monograph. He is an architect and painter, so this text is full of diagrams, paintings, fold-out pages, prose, and indexes.

Willi Egger
ConTEXt has become *the* tool for all typesetting purposes. So in my environment letters, invoices, address–labels, envelope–printing, greeting cards etc are made up in ConTEXt together with functionality provided by Metafun. A favorite issue is making flowcharts for e.g. bookbinding guides, meatprocessing or visualization of the production of a tool. Next to all this I write my own books which are completely made up in ConTEXt including imposition of the pages for folding sections in order to end up with sewn books. E.g. I prepared a 600 pages book which includes over a 120 drawings and metapost-figures. This book is bound as a girdle-book. – Whenever I am asked to give a presentation there is ConTEXt involved.

Gerben Wierda
I have decided not too long ago to move my book-project from LaTEX to ConTEXt (it was my second try, the first one I stopped because there were too many problems). Reason: I dislike how LaTEX output generally looks (though I found memoir to be pretty good) and after reading the documentation I liked how ConTEXt looked as an interface.

Patrick Gundlach

- use it as a reason to run http://contextgarden.net :-)
- Do presentations on it and with it. The tight metapost integration is just great too nice.
- I love to play with new things that I think have a vital future and ConTEXt persuaded me right the first time I saw it on a presentation in Oldenburg or Dortmund (a DANTE meeting).
- prepare University papers and other PDF-based material.

Actually, I do 50% of my typesetting with LaTEX. There are still several things that I can't yet do with ConTEXt (some footnote layout and table typesetting).

I still need some time to generate test documents and ask on the lists for the right way to layout these special cases.

Andrea Valle
I started using LaTEX just not to use MS, and because I was curious. Then I had that idea of creating my scores using one of the LaTEX package. Good idea but nothing worked (namespace collisions, e.g.). So I found ConTEXt, fully equipped with Metafun. The chic solution for my A2 scores with PDF graphics.

Actually I use ConTEXt:
i) as my algorithmic musical composition typesetting environment
ii) for my university courses' presentations
iii) as an output device for all text-based communication.

Matthias Weber
I use typesetting mainly for math: papers, class notes and slides. I started with Signum (a wysiwyg program for the Atari, which was great), switched to AmsTEX, then to LaTEX, and now to ConTEXt. I am bad at remembering macros, I wrote most papers using cut & paste, and when I needed to do something in LaTEX that I couldn't look up in a previous paper, I spent lots of time digging through the 1001 macro packages available for LaTEX.

Mikael Persson
I am mostly playing around a bit. However, since I work with math it is mostly something that has to do with math. Some examples:

- Slides from a conference talk;
- Notes to my students;
- A "second-rental" contract;
- Some playing with the Mathematica fonts;
- Some sudokus;
- Melodikrysset;
- A crossword for a friend;
- Love letter paper;

Olivier Billet
I use it over plain TeX for mainly these reasons:

- ease of type1/otf font set up
- ability to easily deal with layers
- tight connection with PDF file format (like interactive features)

I also thought learning a new macro package was worthwhile because of the support for XML file format will enable to connect TEX with a lot of other appplications.

Up to now, I used it for typesetting my thesis and conference slides.

Henning Hraban Ramm
There are two books that I typeset with ConTEXt:

- a collection of Kofi Annan's speeches
- a portrait of and story collection for German Unitarian religious fellowship

Personally I use ConTEXt for PDF/paper output of my planner including address database, calendar and songbook. (Addresses are stored in a database, collected with a script and written to a ConTEXt table; calendar data including moon phases is computed, combined with events, holidays, personal appointments etc. from several lists and written to ConTEXt files by another script; the songbook contains LilyPond scores in ConTEXt files.)

Steffen Wolfrum
We are using ConTEXt for typesetting wonderful books ("werksatz"), mostly in the fields of arts and humanities. This kind of books often show an advanced use of footnotes, languages and alphabets.

Idris Samawi Hamid
Present activities: I use ConTEXt to
1. typeset an academic journal
2. typeset my own books

Planned or in-progress activities: I am striving for
1. advanced critical edition support
2. advanced Arabic-script support, including complicated scripts like Nastaliq
3. related to 2) I would like to see an integration of aleph and pdfetex (for full left-to-right and right-to-left typesetting)

Xiao Jianfeng
I use ConTEXt to typeset:
1. slides
2. mathematical formula
3. Chinese

It is far easier to write slides in ConTEXt than in LaTEX. And ConTEXt can produce PDF directly.

The math environment is not as good as in LaTEX, but it is enough for me now.

The Chinese environment is not as good as LaTEX(CJK) either, but it is easier to learn. I hope the Chinese module can be enhanced some day.

Tobias Burnus
As big project I typeset my Diploma ($\approx$ Master) thesis with it – and I will do so for my PhD as well. Otherwise I use it for slides and writeups. For papers I so far used LaTEX, but I'm toying with the idea to use it also for a paper; unfortunately APS and arXiv.org don't support ConTEXt. However, if enough authors bug them, they will probably do so at the end. ;-)

Peter Münster
My first experience with ConTEXt was typesetting my brother's PhD thesis in chemistry. Used features: modes (A5 and A4), project-structure, a lot of floats, bibliography.

ConTEXt at work:
□ technical specifications for electronic devices
□ presentations
□ documentation
□ reports

ConTEXt at home:
□ construction plans (one page per figure with well defined scale)
□ letters
□ finding solutions for problems with ConTEXt at work (with the help of the mailing-list)

Hans van der Meer
I use ConTEXt for the presentations in my lectures on Cryptography at the University of Amsterdam. I am considering changing my lecture notes (some 300 pages) from LaTEX into ConTEXt, especially because the ease of incorporating MetaFont pictures into the text. Also for small documents as for example cd-covers.

Charles Doherty
I use ConTEXt for
1. my class guides (incorporating genealogical tables and other graphics) and book-lists in both paper and screen versions
2. for typesetting the Newsletter of the Group for the Study of Irish Historic Settlement
3. my daughter used ConteXt (with my help) to typeset her thesis
4. I have given three classes in the use of ConTEXt to my postgraduate students urging them to use ConTEXt to typeset their theses. Together with Omnigraffle and Bibdesk ConTEXt is ideal for students writing theses. I have been using XeTeX more and more for the ease of getting diacritical marks (when transliterating Sanskrit for example) or medieval characters using the SIL font Gentium.

Alan Bowen
One of the key features of ConTEXt for me is its ability to create platform independent output that combines the ancient languages and mathematics in ways typical of publications in the humanities. So, first, my heartiest thanks to you both and also to Thomas Schmitz for his Greek module.

I currently use ConTEXt to publish the Institute's review/journal (which is available on line, in print, and on CD), as well as its brochures and announcements. I will also be using it to produce its books too.

I have designed a book publication series for my brother using ConTEXt; and I use it to write my own papers, reviews, and books.

Ciro Soto
I wrote my novel using ConTEXt. Inside figures with metapost, the cover layout with Scribus.
See http://www.TheGuitarMakerExploration.com

Olivier Turlier
I started informatics at 33 when coming back to studies: 2 years later i've done my thesis with LaTEX (took me 2.5 years!). Now i try to teach building technics and design, with the help of paper and electronic courses made with ConTEXt. Hopefully, my boss doesn't know that it took me sometimes 2 days for writing a one morning course!

I must recognise that sometimes I think of having a good dtp software, but after some test, I come back to ConTEXt. In the next decades, I'll try to succeed in quiz making and/or nice looking book. I wish to meet people using Context one day.

Bernd Militzer
I typeset examinations and tests for my wife's students at school. For that task I wrote based on exam.cls (LaTEX) a ConTEXt-module.

All private and business letters are typeset with my special DIN-Brief-Modul, based on akletter.cls (LaTEX) and DIN-letter (ConTEXt) from Holger Schöner.

I typeset different articles for my father.

I typeset a book (350 page) with lots of tables, german and many czech words.

My next project is a book about my family from 1600 to now.

Thomas A. Schmitz
I'm a scholar in Classics, and I mostly use ConTEXt

- to write my own lecture notes and typeset them on index cards,
- to prepare my screen presentations,
- for all kind of course-related materials;
- and I'm currently preparing a book that will be typeset with ConTEXt.

Frank Sonnemans
After using LaTEX I use ConTEXt in my work environment (MS office based) for:

- Letters, Quotes and Faxes
- Reports (with floats and references)
- Price lists using perl based conversion of a database export

But I must admit that I now increasingly use Apple Pages to write letters, faxes and the like as it just works faster (copy/paste graphic for example). For large documents ConTEXt stays the preferred option.

Otared Kavian
I am a mathematician and I use ConTEXt mainly for my conferences when I do a presentation on computer.

Since I am a Mac user, it is possible to use XeTeX for writing Right-to-Left material and profit from (almost) all ConTEXt features (many thanks, among others, to Hans Hagen, Johnaton Kew, Adam Lindsay, and Gerben Wierda who made this possible). I use XeTeX + ConTEXt for writing Persian (or Faarsi), which is a non-Arabic language using Arabic alphabet.

Recently I use also ConTEXt for some lecture notes in mathematics for my students, but unfortunately I gave up learning enough of ConTEXt capacities to write a book with it... For this still I use plain TEX and some personal macros, since it seems easier to adapt them to the publisher's constraints.

Volker RW Schaa
I developed a workflow for (particle accelerator) conference proceedings using XML, Perl, and LaTEX. This task was an easy one, because it doesn't matter whether you need to combine 50 or 1400 papers, each page or book will look exactly the same for each conference series.

But I became very dissatisfied with the standard abstract booklet(s) which have to be produced for the participants of such conferences. Here you (nearly) have the full freedom of putting the information together. When my original design (using bleeding boxes for titles, author names, and paper code information) brought me to the limits of LaTEX, I discussed what I wanted with Hans. His statement, 'No problem!', led to the question 'How?'. Within a week I learnt basic ConTEXt, and Hans provided the macros to combine METAPOST and TEX in a way that still leads to questions 'How was that done?' So I now always include a 'Production Note' in my books and booklets to tell people what you can do with ConTEXt and TEX.

Jose Antonio Rodriguez
Til now I used to typeset/illustrate books with DSP tools and my aim is use ConTEXt/MetaFun to do it. Right now I'm converting html based documentation through the xml toolchain (texml), but most of the time I fight with some obscure parameter trying to put things where I want. Integrating MetaPost is definitely a plus for a designer.

Tom Fossen
I use ConTEXt for making the church-magazine of the parish of which I am the minister. With Metafun I constructed headers, page-numbers and all sorts of symbols - forming a style that connects all sorts of input from members and bodies within the community. It's great fun to see each monthly edition grow, from (sometimes) scraps of paper into an well-organized and good looking magazine, complete with photos and other illustrations.

I learned using ConTEXt while writing papers for my training as a contextual therapist. And since I started a practice as contextual therapist I use it for flyers, correspondence, bills.

Vit Zyka

- proceedings from individual PDFs,
- presentations,
- posters,
- logos, visiting cards,
- manual (HTML and PDF form from XML source),
- personal electronic travel-journal
- booklet with very complex layout (sometimes seems to me over my ConTEXt skills) with two-column, margin, many, many floats, balancing, sorting, in Czech, ... I will write some experience with this – if I will finish it and survive two times postponed deadline...

Lutz Haseloff

- □ I write all my personal documents with ConTEXt
- □ I prepared some complicated fillable Forms including calculations by Javascript and sending the data to a web server
- □ I convert some Manuals to ConTEXt (step by step)
- □ I query an Oracle database with perl and write the results to ConTEXt files
- □ all my presentations I create with ConTEXt
- □ I typeset chinese texts

Jörg Hagmann

As the editor of the 70+ page annual report of an ornithological society, I learned to hate word. When I was approached by a publisher with the offer of writing a textbook of biochemistry, I decided to try something more suitable and chose – just why is hard to say, I didn't know anything about any of the possibilities – ConTEXt. It was very hard simply to find out what it was and then how to install it and after that how to get some presentable result. But it's great – thanks to everybody.

It is always claimed that writing in a text editor and leaving the layout/typesetting part for later is better because you concentrate on writing. This is not true, at least not when working with ConTEXt. It is much more fun to "tex" what you have, to try out this and that, than to think about the next sentence. I now write with a Parker Duofold (bought years ago while on a bicycle tour in the Netherlands!) in a notebook before I go to the computer.

Mark Pearson

I'm currently working in a team which is building a system for the automatic production of office stationery (business cards, letterheads and compliment slips) with ConTEXt at its heart.

The system provides a web-application front end for selected users to create stationery orders. These orders are then processed to produce .tex files, which are processed by ConTEXt to produce press-ready PDFs. The PDFs are sent with order details straight to our inhouse printing department.

Nicolas Grilly

What I do with ConTEXt:

1. Produce sales and marketing reports for my customers (we are a marketing and data mining consultancy). We have a very simple templating language written in Python that connects to an SQL database, generates the ConTEXt file, launches ConTEXt and launches Acrobat Reader to see the PDF file.
2. Produce totally personalized mailings for customers of companies we work for. Everything is personalized: text, background, photos, etc. We produce it with the same templating solution (see previous point).

Taco Hoekwater (editor)

Een uittreksel uit de recente bijdragen in het CTAN archief

Abstract
Dit artikel beschrijft een aantal recente bijdragen uit het CTAN archief (en andere bronnen op het Internet). De selectie is gebaseerd op wat ik zelf interessant vind en wat ik denk dat voor veel anderen interessant is. Het is dus een persoonlijke keuze. Het heeft niet de bedoeling om een volledig overzicht te geven. De uitgebreidere bijdragen zijn ook geen handleidingen. Beschouw het maar als een soort menukaart die de bedoeling heeft om de lezer lekker te maken.

Keywords
TEX, LaTEX, ConTEXt, packages, CTAN, classes, graphics, programma's.

Inleiding

Sinds de laatste aflevering heb ik nogal wat zien langskomen aan CTAN aankondigingen. Een grove schatting is zo'n 350 berichten. Sommige pakketten staan er overigens meerdere keren in. Kennelijk zijn enkele auteurs erg actief, met soms wekelijkse updates.

Mijn CTAN-mailbox begint langzamerhand wel erg groot te worden: op dit moment 2737 berichten vanaf 1994. Ik probeer wel verouderde berichten weg te gooien, maar het is veel werk om uit te zoeken welke berichten vervangen zijn door nieuwere, en soms staat in oude berichten ook nog interessante informatie.

Deze keer ligt de nadruk op grafische mogelijkheden en speciale toepassingen (scheikunde, muziek, sudoku en dergelijke).

Grafische pakketten

Een van de dingen die me opviel bij het doorlezen van de CTAN-aankondigingen was dat er de laatste tijd veel nieuws was op grafisch gebied. Terwijl TeX zelf geen grafische voorzieningen had en heeft, wordt dit kennelijk een steeds belangrijker aspect van het maken van documenten. Maar met het gebruik van het `special`-commando en de extra mogelijkheden die PDFTEX biedt wordt het invoegen van grafisch materiaal steeds gemakkelijker. Op zich zijn er niet zoveel nieuwe voorzieningen hiervoor gekomen maar er zijn veel updates van de bestaande pakketten. Mijns inziens toont dit aan dat TEX nog lang niet dood is.

We kunnen de bijdragen indelen in een paar verschillende categorieën.

Programma's

Afzonderlijke programma's (dus los van TEX zelf) kunnen gebruikt worden aan de voorkant of aan de achterkant (en soms tijdens een TEX-run).

fig2vect Fig2vect is een programma om Fig bestanden om te zetten naar een formaat dat interessant is voor gebruik in TEX. Fig bestanden worden geproduceerd door en bewerkt met het Unix-programma XFig. (Het Fig formaat is een simpel ASCII formaat dat ook gemakkelijk door programma's gegenereerd kan worden.) Op Unix-systemen is dit een populair tekenprogramma. Vergeleken met programma's als Adobe Illustrator heeft het niet zoveel functionaliteit, maar voor het tekenen van schema's (met blokken, cirkels, lijntjes, tekst en degelijke) vind ik het zelf een prettig programma. De gebruikersinterface is een beetje ongewoon. In andere tekenprogramma's is het gebruikelijk dat je eerst een object uitkiest waarop je een bewerking wilt uitvoeren en dan klik je op een icoon of selecteer je een menu-item om de bewerking te doen. In XFig daarentegen kies je eerst de bewerking en daarna klik je op een object waarop de bewerking uitgevoerd moet worden. Bij het tekenen van lijnen, rechthoeken en dergelijke moet je bij de meeste programma's eerst op het beginpunt klikken en dan slepen naar het eindpunt resp. ander punt en dan de muisknop loslaten. Bij XFig daarentegen moet je het eindpunt ook met een muisklik aangeven. Eenmaal hieraan gewend werkt dit best prettig maar als je regelmatig moet wisselen tussen XFig en andere programma's kan het wel moeilijk zijn. Intussen bestaan ook klonen van XFig voor Windows: jFig (in Java geschreven, dus platformonafhankelijk) en WinFig. Het is een tijd geleden dat ik met jFig gewerkt heb; in die tijd zat er een aantal bugs in de gebruikersinterface. WinFig wordt door de auteur uitdrukkelijk niet een kloon genoemd, maar is een op XFig geïnspireerd programma dat specifiek gebruik maakt van de Windows functionaliteit. Het kan echter Fig files lezen en schrijven. jFig en WinFig zijn beide shareware programma's. XFig is gratis.

Alle drie de programma's hebben mogelijkheden om naar andere formaten te exporteren, bijvoorbeeld naar CGM, EMF, epic, GIF, HPGL, JPEG, LaTeX, MetaFont, MetaPost, PCX, PDF, PNG, (Encapsulated)PostScript, pstex, tk, SVG formaten (de SVG export van XFig is nog niet erg volwassen). XFig gebruikt een extern programma om de conversie te doen (als grafische gebruiker merk je dat overigens niet); dit programma heet fig2dev. Fig2vect is nu een alternatief programma voor sommige van deze conversies, namelijk naar MetaPost, EPS, PDF, TeX en SVG.
Waarom fig2vect?

□ fig2vect is speciaal geschreven voor gebruik met LaTeX of PDFLaTeX.
□ X-Splines (dit is een nieuw soort geometrische constructie die XFig gebruikt voor krommen in tekeningen) worden naar Bezier splines (dit is wat de meeste grafische programma's gebruiken) omgezet. In fig2dev worden ze omgezet in een serie lijnsegmentjes.
□ Vulpatronen worden als vectorelementen geïmplementeerd in plaats van bitmaps.
□ Voor configuratie gebruikt het een bestand in plaats van commandline argumenten.

Fig2vect is te vinden op in CTAN op `/support/fig2vect`. XFig op `http://www.xfig.org/`, jFig op `http://tech-www.informatik.uni-hamburg.de/applets/jfig/` en WinFig op `http://www.schmidt-web-berlin.de/WinFIG.htm`.

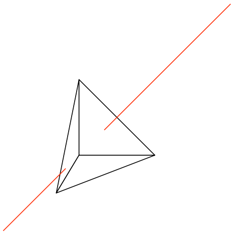
LaTeXPiX LaTeXPiX is een tekenprogramma voor gebruik onder Windows dat LaTeX code genereert. Het ondersteunt zowel eepic als pgf. Deze LaTeX pakketten hebben meer mogelijkheden dan de standaard LaTeX picture omgeving. Over pgf zie meer hieronder. Enkele mogelijkheden van LaTeXPiX:

□ Gebruik van kleur voor alle objecten.
□ Opvullen van objecten met kleur of grijstinten.
□ Tekst met vet en cursief.
□ Invoegen van plaatjes in JPEG of PNG formaat.
□ Pijlen in elke richting.
□ Elliptische bogen.
□ Splines (krommen).

sketch Sketch is een programma dat tekeningen gedefinieerd in een eigen taaltje omzet in Pstricks-code die daarna in een TeX-document ingevoegd kan worden. Het benadrukt vooral 3-dimensionale objecten. Een voorbeeld:

```
% vertices of the tetrahedron
def p1 (0,0,1) def p2 (1,0,0)
def p3 (0,1,0) def p4 (-.3,-.5,-.8)
% faces of the tetrahedron.
polygon(p1)(p2)(p3) % original front polygon
polygon(p1)(p4)(p2) % bottom
polygon(p1)(p3)(p4) % left
polygon(p3)(p2)(p4) % rear
% line to pierce the tetrahedron
line[linecolor=red](-1,-1,-1)(2,2,2)
```

Dit levert dit plaatje op:



Het pakket is te vinden op CTAN onder `/graphics/sketch` of op `http://www.frontiernet.net/~/eugene.ressler`.

png2pdf Het programma png2pdf is ook al eerder genoemd in deze rubriek. Het vertaalt een PNG plaatje naar PDF waarbij transparantie behouden blijft. Alleen de documentatie is deze keer bijgewerkt. Zie op CTAN onder `/support/png2pdf`.

dvisvgm Dvisvgm is een programma om DVI files om te zetten naar het Scalable Vector Graphics (SVG) formaat. Op CTAN: `dviware/dvisvgm`.

dvipng Dvipng zet DVI files om naar plaatjes in het PNG formaat. Op CTAN: `/dviware/dvipng`. De nieuwe versie ondersteunt alpha channel (transparantie) support and het pakket xcolor.

dvi2bitmap Dvi2bitmap is een programma om DVI-bestanden rechtstreeks om te zetten in bitmaps van het formaat XBM. GIF en PNG. Dus zonder tussenkomst van Postscript. Dit betekent natuurlijk ook dat geen EPS plaatjes ingevoegd kunnen worden, maar het is vooral bedoeld om bij de conversie van TeX of LaTeX naar HTML formules en dergelijke te kunnen omzetten in bitmaps.
Op CTAN: `/dviware/dvi2bitmap`.

Macropakketten

PGF Over pgf heb ik de vorige keer al geschreven, maar na het kijken in het manual (bijna 250 pagina's!) vond ik het zo indrukwekkend dat het de moeite waard is om er meer over te schrijven. Er is intussen trouwens weer een nieuwe release uitgekomen dus dat is op zichzelf al genoeg reden. Als je het manual doorbladert (voor echt doorlezen heb ik nog niet genoeg tijd gehad) dan straalt de Duitse Gründlichkeit er vanaf. Hier heeft iemand zich uitgeleefd om iets goeds te maken. Als je bedenkt dat dit allemaal met behulp van TEX-macro's is geprogrammeerd dan word je even stil. Pgf wordt uitbundig gebruikt in het beamer pakket voor presentaties.

Pgf is een pakket, geschreven door Till Tantau, dat zowel Postscript (door middel van dvips) als PDF (bij gebruik van PDFTEX) kan genereren. Dit in tegenstelling tot bijvoorbeeld Pstricks dat alleen Postscript ondersteunt. Pgf is zowel met LaTEX als met plain TEX (en daardoor ook met ConTEXt) te gebruiken. Behalve Postscript en PDF kan het pakket ook SVG produceren: dit is voor gebruik met het programma tex4ht. Dit laatste genereert HTML vanuit een LaTEX document. Door nu voor de pgf-plaatjes SVG te genereren krijg je een HTML bestand met vector-plaatjes in plaats van bitmaps.

Pgf is in verschillende lagen opgebouwd. De basiscommando's kunnen verschillende soorten objecten genereren, zoals lijnen, veelhoeken, bogen, krommen en tekst. Het kan zowel de objecten als de vullingen in kleur doen, en gebruikt hiervoor het xcolor pakket. Ook zijn kleurovergangen mogelijk en dit wordt bijvoorbeeld gebruikt voor schaduwrandjes.

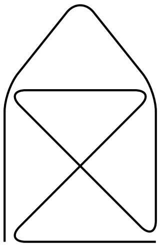
Bovenop de laag met basiscommando's zijn verschillende uitgebreide toepassingen aanwezig.

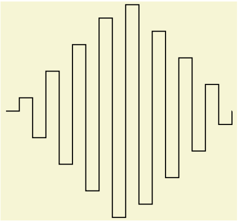
□ Nodes zijn stukken tekening die op een bepaalde plaats gezet en met elkaar verbonden kunnen worden.
□ Boomstructuren zijn een toepassing van nodes.
□ 'Snakes' zijn paden met een zigzagstructuur
□ Plotten van functies met parabolen en sinussen.
□ Transformaties van objecten zoals roteren en vergroten.
□ Clippen van tekeningen.
□ Transparante teksten.

TikZ ('TikZ ist *kein* Zeichenprogramm') is een verzameling macro's die een tekentaal vormen die op pstricks en Metafont is geïnspireerd. Een voorbeeld:

```
\tikz \draw[thick,rounded corners=8pt]
(0,0) -- (0,2) -- (1,3.25) -- (2,2) -- (2,0)
  -- (0,2) -- (2,2) -- (0,0) -- (2,0);
```

Dit levert het linkerplaatje op, terwijl rechts een voorbeeld van een snake staat.





Er is ook een conversieprogramma van Fig naar pgf geschreven door Thomas Neumann.

PGF kan gevonden worden op CTAN onder /graphics/pgf/ en op http://sourceforge.net/projects/pgf/ (waar ook een link naar het conversieprogramma van Fig naar pgf staat).

pdf-trans Pdf-trans is verzameling macros voor transformaties van TEX-boxen (gebaseerd op on plain and PDFeTEX). Op CTAN /macros/generic/pdf-trans/.

movie15 Movie15 is een macro-pakket om multimedia-elementen in een PDF-bestand te krijgen. In tegenstelling tot wat de naam suggereert gaat het hierbij niet alleen om filmpjes, maar bijvoorbeeld ook over geluid. Bijvoorbeeld:

```
\includemovie[autoplay]{0pt}{0pt}{pagesound.mp3}
```

kan gebruikt worden om een geluid af te spelen.

Pstricks De ontwikkeling van Pstricks gaat gewoon door. Onder andere de volgende onderdelen zijn in nieuwe versies uitgekomen:

□ pst-circ voor het tekenen van electrische schakelingen
□ pst-func voor het plotten van speciale mathematische functies
□ pst-text heeft ondersteuning voor 16-bits Japanse tekens
□ pstricks zelf had vroeger eigen code voor kleurgebruik die afweek van het standaardpakket color (en de uitbreiding xcolor). De eigen code stond in pstcol.sty. De nieuwe versie gebruikt nu het color of xcolor pakket, en pstcol.sty is overbodig geworden. Het bestaat echter nog wel in een gewijzigde vorm die de standaard kleurondersteuning uitschakelt.

pst-pdf Een belangrijke beperking van het gebruik van pstricks is dat het echte Postscript code genereert. Omdat Postscript een complete programmeertaal is heb je hiervoor een compiler of interpreter nodig die die taal begrijpt (voor zover ik weet bestaat er geen compiler voor). De bekendste interpreter is waarschijnlijk wel Ghostscript. Verder bevat elke Postscriptprinter ook zo'n interpreter.

Bij gebruik van PDFTEX ontbreekt zo'n interpreter en zonder trucs is pstricks dus niet met PDFTEX te gebruiken. De trucs komen er in het algemeen op neer dat een deel van het document via de route LaTEX–dvips–Ghostscript omgezet wordt naar PDF-plaatjes, waarna het document door PDFLaTEX met de gegenereerde PDF-plaatjes in een PDF-bestand omgezet wordt. De vorige keer heb ik de pakketten pdftricks en ps4pdf besproken waarmee dit kan. Deze zijn vrij omslachtig omdat de stukken waar Postscript in staat in een psinputs omgeving gezet moeten worden, inclusief die in de preambule, bijvoorbeeld

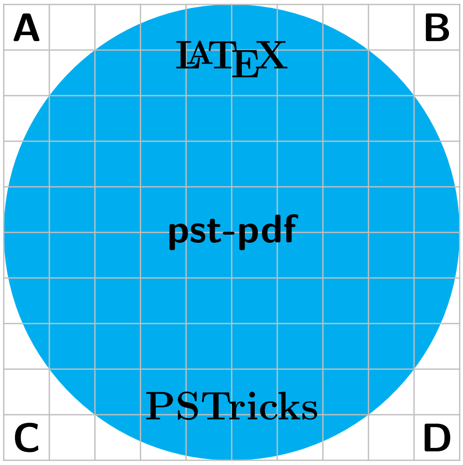
```
  \begin{psinputs}
    \usepackage{pstricks}
  \end{psinputs}
  ...
  \begin{pspicture}(5,2)
    \psline{|<->|}(0,0.3)(4,1.9)
  \end{pspicture}
```

Dit moet liefst nog conditioneel zodat de code niet uitgevoerd wordt als je gewoon LaTeX gebruikt:

```
\usepackage{ifpdf}
\ifpdf%
  \usepackage{pdftricks}
  \begin{psinputs}
    \usepackage{pstricks}
  \end{psinputs}
\else
  \usepackage{pstricks}
  \newenvironment{pdfpic}{}{}
\fi
...
\begin{pdfpic}
  \begin{pspicture}(5,2)
    \psline{|<->|}(0,0.3)(4,1.9)
  \end{pspicture}
\end{pdfpic}
```

Pst-pdf is een vervanging van het pakket ps4pdf waarmee dit gemakkelijker gaat. Het is voldoende om `\usepackage{pst-pdf}` in te voegen en het bestand kan zowel met gewoon LaTeX als met PDFLaTeX gebruikt worden. In het laatste geval moet dus eerst een run gedaan worden met LaTeX, dvips en ps2pdf, maar hiervoor is een script ps4pdf aanwezig. Dit pakket is te vinden op CTAN: `/graphics/pstricks/contrib/pst-pdf/` of op `http://perce.de/LaTeX/pst-pdf/` . Hierbij een voorbeeld:

```
\documentclass[12pt]{article}
\usepackage{pstricks}
\usepackage{pst-pdf}
\pagestyle{empty}
\begin{document}

\begin{pspicture}(-5.25,-5.25)(5.25,5.25)%
  \pscircle*[linecolor=cyan]{5}
  \psgrid[subgriddiv=0,
          gridcolor=lightgray,
          gridlabels=0pt]
  \Huge\sffamily\bfseries
  \rput(-4.5,4.5){A} \rput(4.5,4.5){B}
  \rput(-4.5,-4.5){C}\rput(4.5,-4.5){D}
  \rput(0,0){pst-pdf}
  \rmfamily
  \rput(0,-3.8){PSTricks}
  \rput(0,3.8){\LaTeX}
\end{pspicture}

\end{document}
```



Speciale toepassingen

Macropakketten

mhchem Mhchem is een LaTeX pakket voor het zetten van chemische formules en vergelijkingen. Sub- en superscripts met de normale math-mode van TeX staan vaak op de verkeerde hoogte. Bovendien maakt math-mode de letters cursief, terwijl ze in chemische formules rechtop moten zijn. Dit pakket maakt het makkelijk om formules en vergelijkingen te zetten omdat een simpele notatie gebruikt wordt. Bijvoorbeeld `\ce{1/2H2O}` levert op: $\frac{1}{2}\mathrm{H_2O}$. het commando `\cee` is voor vergelijkingen. Het accepteert `&` en `\\` als in tabular- en array-omgevingen. Reactiepijlen kunnen met `->` aangegeven worden. In deze versie is de syntax veranderd maar met de pakket-optie `version=2` kan de oude syntax gebruikt worden. Er is ook nog een pakket rsphrase voor de officiële 'Risk and Safety' uitdrukkingen voor chemische stoffen in het Engels, Deens, Duits en Frans.
De pakketten zijn te vinden op CTAN onder `/macros/latex/contrib/mhchem/`.

harmony Een LaTeX pakket voor het zetten van harmonie-symbolen in musicologische teksten. Op CTAN onder `/macros/latex/contrib/harmony`.

TeXmuse TeXmuse is een verzameling macro's om professionele muziek te zetten met behulp van TeX en Metafont. Dit is een eerste versie, dus nog behoorlijk beperkt. De auteur heeft het gebruikt voor het zetten van Bach's inventions en dergelijke. Op CTAN: `/macros/texmuse/`.

byzfonts Dit zijn fonts voor het zetten van Byzantijnse muziek (de officiële muziek van de Grieks-Orthodoxe kerk). Op CTAN: `/fonts/byzfonts/`.

bibleref Dit pakket kwam voort uit een discussie op de nieuwsgroep comp.text.tex over het citeren van referenties naar bijbelteksten in LaTEX. Het probleem was om op een consistente manier te citeren, bijvoorbeeld met de namen van de bijbelboeken al dan niet afgekort. Dit pakket geeft hiervoor ondersteuning. Te vinden op CTAN: `/macros/latex/contrib/bibleref/`.

maltese Een pakket voor het invoeren van Maltese lettertekens (ċ, Ċ, ġ, Ġ, ħ, Ħ, għ, Għ, ie, Ie, ż, Ż) in LaTEX is te vinden op CTAN onder `/language/maltese/`. Dit is uiterst nuttig nu Malta lid is geworden van de EU.

floatrow Een LaTEX pakket om de layout van floating omgevingen te configureren. Bijvoorbeeld om floats naast elkaar te zetten of de caption naast de float te zetten. Dit pakket kan samenwerken met het caption pakket (versie 3.x). Op CTAN: `/macros/latex/contrib/floatrow/`.

pdfcolmk PdfTEX heeft van zichzelf geen goede ondersteuning voor kleur. Het is wel mogelijk kleur te gebruiken, maar bij pagina-overgangen kan het mis gaan. Dit komt omdat er geen kleur-*stack* is. Bij een pagina-overgang kunnen er verschillende kleuren een rol spelen: die van de lopende tekst, de headers en footers en eventueel een voetnoot die misschien ook nog over de paginagrens heen gaat. Verder kan het ook nog mis gaan bij floats en marginpars. Het pakket pdfcolmk probeert al deze situaties op te lossen met behulp van marks. Op CTAN: `/macros/latex/contrib/oberdiek/pdfcolmk.sty`.

flowfram Dit pakket is bedoeld om een document te maken dat bestaat uit frames waarbij de tekst van het ene frame overloopt in een ander frame. Dit is nuttig voor posters, brochures, tijdschriften en dergelijke die zich niet zo makkelijk in het formaat van één of twee kolommen laten wringen.
Te vinden op CTAN: `/macros/latex/contrib/flowfram/`.

tabularht Het LaTEX pakket tabularht bevat een aantal omgevingen die varianten zijn van tabular en array met een extra parameter die de hoogte specificeert. De omgevingen zijn tabularht, tabularht* en arrayht. Als het pakket tabularx gebruikt wordt dan komen hier nog bij tabularxht en tabularxht*
De nieuwe versie heeft een optie 'vlines' die probeert het probleem van onderbroken verticale lijnen op te lossen.
Op CTAN: `/macros/latex/contrib/oberdiek/tabularht.sty`

polytable Polytable implementeert een omgeving die vergelijkbaar is met tabular waarbij kolommen namen kunnen krijgen en entries tussen de kolommen geplaats kunnen worden. Dit is onder andere heel handig voor programmacode die uitgelijnd moet worden. Hier is het ook speciaal voor ontwikkeld. Het is te vinden op CTAN onder `/macros/latex/contrib/polytable/`. Hier volgt een voorbeeld van zo'n tabel:

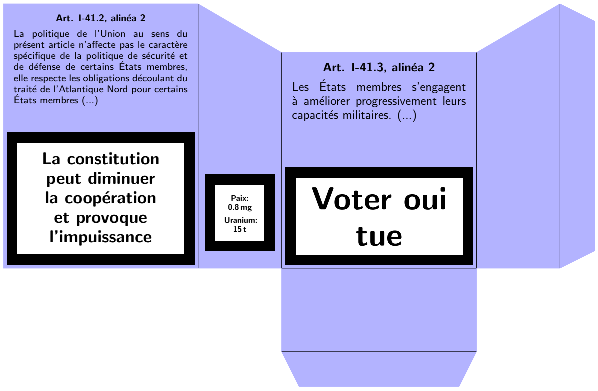
class (Eq a) ⇒ Ord a **where**

$$\begin{array}{lll}
\mathit{compare} & :: & a \to a \to \mathsf{Ordering} \\
(<), (\leq), (\geq), (>) & :: & a \to a \to \mathsf{Bool} \\
\mathit{max}, \mathit{min} & :: & a \to a \to \mathsf{Bool}
\end{array}$$

— Minimal complete definition: ($\leq$) or *compare*
— using *compare* can be more efficient for complex types

$$\begin{array}{llll}
\mathit{compare}\ x\ y & \mid x \equiv y & = & \mathsf{EQ} \\
 & \mid x \leq y & = & \mathsf{LT} \\
 & \mid \mathit{otherwise} & = & \mathsf{GT} \\
x \leq y & & = & \mathit{compare}\ x\ y \not\equiv \mathsf{GT} \\
x < y & & = & \mathit{compare}\ x\ y \equiv \mathsf{LT} \\
x \geq y & & = & \mathit{compare}\ x\ y \not\equiv \mathsf{LT} \\
x > y & & = & \mathit{compare}\ x\ y \equiv \mathsf{GT} \\
\mathit{max}\ x\ y & \mid x \leq y & = & y \\
 & \mid \mathit{otherwise} & = & x \\
\mathit{min}\ x\ y & \mid x \leq y & = & x \\
 & \mid \mathit{otherwise} & = & y
\end{array}$$

Zigarettenschachtelhuellenzeichnung Dit pakket mag wel de prijs voor de mooiste naam krijgen. Het is bedoeld om sigarettendoosjes (of beter gezegd overtrekken voor deze doosjes) te maken waar de officiële waarschuwingen vervangen zijn door grappige teksten. Heel nutig voor degenen die hun verslaving belangrijker vinden dan hun gezondheid. Op CTAN: `/graphics/zigaretten/`. Een voorbeeld:



Art. I-41.2, alinéa 2

La politique de l'Union au sens du présent article n'affecte pas le caractère spécifique de la politique de sécurité et de défense de certains États membres, elle respecte les obligations découlant du traité de l'Atlantique Nord pour certains États membres (...)

La constitution peut diminuer la coopération et provoque l'impuissance

Paix: 0.8 mg
Uranium: 15 t

Art. I-41.3, alinéa 2

Les États membres s'engagent à améliorer progressivement leurs capacités militaires. (...)

Voter oui tue

sudoku Met dit pakket kun je sudoku-puzzels tekenen. Je vindt het op CTAN onder `/macros/latex/contrib/sudoku/`. Degenen die de laatste jaren op een andere planeet hebben doorgebracht en niet weten wat sudoku is, is meer informatie te vinden op `http://www.sudoku.org.uk`. Zie hieronder een voorbeeld.

```
\setlength{\sudokusize}{\linewidth}
\begin{sudoku}
|2|5| | |3| |9| |1|.
| |1| | | |4| | | |.
|4| |7| | | |2| |8|.
| | |5|2| | | | | |.
| | | | |9|8|1| | |.
| |4| | | |3| | | |.
| | | |3|6| | |7|2|.
| |7| | | | | | |3|.
|9| |3| | | |6| |4|.
\end{sudoku}
```

2	5			3		9		1
	1				4			
4		7				2		8
		5	2					
				9	8	1		
	4				3			
			3	6			7	2
	7							3
9		3				6		4

Programma's

pdfbook Het programma pdfbook kan de pagina's van een PDF-bestand herordenen en eventueel schalen zodat er een nieuw bestand uitkomt dat geschikt is voor het afdrukken van boekjes. Dit is een C-programma dat gebruik maakt van LaTeX en het pakket pdfpages.

fontools Een verzameling programma's voor het bewerken van font-bestanden.

□ afm2afm – reencode .afm files
□ autoinst – om het gebruik van de LCDF TypeTools gemakkelijker te maken.
□ cmap2enc – zet de glyph indices in TrueType fonts om in Adobe glyph namen.
□ font2afm – dit programma creëert font metrieken; het is een script dat de programma's pf2afm, ttf2afm, pfm2kpx and ot2kpx aanroept.
□ ot2kpx – extraheert kerning paren uit een OpenType font.
□ pfm2kpx – extraheert kerning paren uit buggy .pfm files.
□ showglyphs – creëert een PDF bestand waarin alle glyphs van een font staan.

Je kunt deze programma's vinden op CTAN onder `/fonts/utilities/fontools/`.

xpdfopen Twee programma's van Taco Hoekwater (pdfopen en pdfclose) waarmee je de X Window System versie van Adobe's Acrobat Reader kunt openen en sluiten onder Linux. Op CTAN onder `/support/xpdfopen/`.

bibhtml Dit is een Perl script met een bijbehorende BiBTeX style file om een bibliografie te maken voor een verzameling HTML-bestanden. De referenties in de tekst zijn direct gelinkt aan de bibliografische referentie, en als er een URL in de BiBTeX database staat dan bevat de bibliografie deze als link. Op CTAN in `/biblio/bibtex/contrib/bibhtml/`.

TexPoint TexPoint is een programma waarmee LaTeX formules (en andere teksten) in Powerpoint slides kunnen worden ingevoegd. De invoegingen bevatten de originele LaTeX source zodat ze later nog gewijzigd kunnen worden. Deze methode levert fraaiere formules dan de eigen methode van Microsoft Office. Degenen die op de laatste NTG-dag aanwezig waren hebben het effect kunnen zien in de voordracht van Ronald Rietman. De huidige versie van TexPoint werkt op Windows en Mac OS X. Het is te vinden op `http://www.thp.uni-koeln.de/^/ang/texpoint/index.html`.

Documentatie

epslatex Epslatex is al sinds jaar en dag het toonaangevende (gratis) document dat alle ins en outs beschrijft van het invoegen van plaatjes in LaTeX. Er is nu een nieuwe versie (3.0) verschenen die aangepast is voor het gebruik van PDFLaTeX en de nieuwste versies van de pakketten caption en subfigure (tegenwoordig subfig). De auteur (Keith Reckdahl) beschouwt het als een bèta-versie, maar zegt wel dat het een significante verbetering is ten opzichte van de vorige versie. Op CTAN in `/info/epslatex.pdf` (ook in andere formaten beschikbaar).

amslatex Er is een Vietnamese versie verschenen van de gebruikershandleiding van het amslatex pakket. Ik verwacht niet dat wij veel lezers hebben die Vietnamees kennen, maar aangezien PDFTeX Vietnamese wortels heeft wilde ik dit toch vermelden. Op CTAN in `/info/amslatex/vietnamese`.

Piet van Oostrum

Color separation in two-color printing

Abstract
The book *Basisboek wiskunde* ('Basic Mathematics') by Jan van de Craats and Rob Bosch[1] was typeset in LaTeX and submitted for printing as one big pdf-file. In this book one extra color (blue) was used for titles, headings, footings, important formulas, figures and also as a background color for certain pages or parts of text. Jan van de Craats, who did the typesetting, reports on a trick for obtaining color separation without flaws.

Keywords
color, separation, printing

The typescript of our *Basisboek wiskunde* originally was prepared with a black and white text in mind. However, our publisher, Pearson Education Benelux, urged us to use one extra color to enliven the general appearance of the book. Since this publisher has no experience with LaTeX, I had to figure out myself how to include color, of course with the help of Leslie Lamport's LaTeX user's guide[2] and the LaTeX *Graphics Companion*[3].

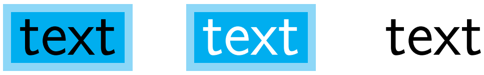


Figure 1. Color separation: full picture, cyan, and black components.

This in general went smoothly, but from a previous book with one extra color for which Wybo Dekker did the final typesetting, I knew that a problem might occur with black text on a colored background when the printing process is not done with the utmost precision. At color separation, two files are produced, one containing the portions to be printed with ink in the chosen color, and the other containing the portions to be printed in black. The latter is always printed last. So, for instance, when black text is to be printed on a blue background, first the blue background is printed *with the black text left white*, and then the text is superimposed (or rather, filled in) in black. See figure 1, where cyan was used instead of our color blue for reasons that will be explained later. This, however, requires a very accurate printing process. If the paper is not aligned properly during the second stage, very thin white 'shadows' might occur along with the black letters. The *Graphics Companion* mentions this problem on page 348 and states that printers solve this by *trapping*, which means that the size of the colored areas is slightly increased. It doesn't say, however, how to achieve this in LaTeX. Before explaining our simple trick that does the job in a different way, let me first devote a few words to color separation in general.

Color separation

Color printing usually is a four step process, each page being printed four times with proportions of the basic colors *cyan*, *magenta*, *yellow* and *black*, respectively. Color separation from a source file consists in producing four distinct files in which the colors are separated so that, e.g., the *cyan* file only contains the proportions of cyan that should be printed. You can do color separation yourself using the `aurora` package together with `dvips`, as described in the *Graphics Companion* [2] on page 349. In this way I learned a lot about the separation process, although finally the color separation for our book was done directly from a (non-separated) pdf source file by the printing company.

In preparing a typescript with only two colors, black and one custom color, it is advisable to use one of the three basic colors cyan, magenta or yellow instead of the custom color. After color separation, the corresponding color file then can be used for printing with ink in the chosen color. In the LaTeX source file for our *Basisboek wiskunde*, I took cyan, although a special kind of blue was used in the final printing. The printing company applied color separation to my file `BasisboekWiskunde.pdf` and used the resulting cyan and black component files for printing with blue and black ink, respectively. The (empty) magenta and yellow component files were left aside.

Shades of color and overprinting

It should be noted that lighter shades of the chosen color can be obtained in much the same way as gray levels can be obtained in printing with black ink. For instance, you might want to use 100 percent blue for some colored text, 40 percent for a medium blue background color and 20 percent for a still lighter kind of blue. In the LaTeX-file, where cyan is used instead of blue, this can be achieved (with the help of the `color` package) by defining colors *fullblue, mediumblue, lightblue* as follows:

```
\definecolor{fullblue}{cmyk}{1,0,0,0}
\definecolor{mediumblue}{cmyk}{.4,0,0,0}
\definecolor{lightblue}{cmyk}{.2,0,0,0}
```

(see also [3], page 132). Shades of color are specified by a number between 0 and 1, with 1 corresponding to full color and 0 to no color at all. Note that we use `definecolor` with `cmyk` as its first argument, specifying the cmyk color system. The four numbers in the second argument of the `definecolor` command correspond to shades of *cyan*, *magenta*, *yellow* and *black*, respectively.

Once you understand the cmyk color system and the principles of color separation, it is not difficult to come up with the following solution to the problem of avoiding white shadows. The trick is to define special colors 'black' corresponding to the chosen shades of background color (*cyan* in our code) as follows:

```
\definecolor{BlackOnFull}{cmyk}{1,0,0,1}
\definecolor{BlackOnMedium}{cmyk}{.4,0,0,1}
\definecolor{BlackOnLight}{cmyk}{.2,0,0,1}
```

Then, during the color separation process, any text that is set in one of these black colors is not only included in the black file, but also in the cyan file with the appropriate shade. Since this shade exactly matches its background color, it becomes invisible, filling completely the spaces that otherwise would have been left white for the letters! At the second (black) printing stage, the black text can be printed safely on this background without fear for white shadows.

For instance, the code

```
\setlength{\fboxrule}{3pt}
\fcolorbox{mediumblue}{fullblue}{%
\textcolor{BlackOnFull}{{\huge text}}}
```

produces a framed box with inner color *fullblue* and a 3 pt frame in *mediumblue* containing 'text' in black. See figure 2, where also the color separation is given.

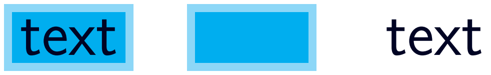


Figure 2. Color separation with our overprinting trick.

It should be noted that our trick works fine since black is used as overprinting color. The color black absorbs all light, even when printed on a colored background. When using a different color for overprinting, the two colors might interfere, and our trick might produce undesired effects. I have no experience in this respect, and would welcome any other solution.

References

[1] Jan van de Craats and Rob Bosch. *Basisboek Wiskunde*. Pearson Education Benelux (2005). ISBN 90-430-1156-8.

[2] Leslie Lamport. *LaTeX: A Document Preparation System. User's Guide and Reference Manual*. Addison-Wesley, 2nd edition (1994). ISBN 0-201-52983-1.

[3] Michel Goossens, Sebastian Rahtz, and Frank Mittelbach. *The LaTeX Graphics Companion*. Addison-Wesley (1997). ISBN 0-201-85469-4.

Jan van de Craats
`janvandecraats@casema.nl`

powerdot

making presentations with LaTEX

Abstract
This article describes some technical details of the powerdot class [3] which was developed during the summer holidays of 2005.

Keywords
LaTEX presentations, powerdot, prosper, pstricks, xkeyval

Introduction

powerdot is a presentation class for LaTEX that allows for the quick and easy development of professional presentations. It comes with many tools that enhance presentations and aid the presenter. Examples are automatic overlays, personal notes and a handout mode. To view a presentation, DVI, PS or PDF output can be used. A powerful template system is available to easily develop new styles. Also, a LYX layout file is provided. powerdot is a new package in the line of prosper [5] and HA-prosper [1].

It has been well known, for quite some time, that the prosper class has severe problems. Examples include damaged constructions from a redefined `\item`, spacing problems on overlays while in math mode, failing counter protection, useless DVI and PS output, and a lack of support for screen-optimized paper dimensions. The HA-prosper package tried to correct some of these problems, but as LaTEX programming experience grew, it was found that some of the problems of the prosper class (such as the paper job) could not be corrected anymore.

However, the idea of using pstricks [6, 7] and `minipage` environments for content was appealing in that it allowed for a vast variety of presentation styles.

Halfway through 2004, Hendri decided to make a successor to the prosper and HA-prosper combination. The class would be built from the ground up, and it would be called powerdot.[1] As it would be a major undertaking to develop a new class, new styles, and documentation, Hendri looked for a helping hand on the HA-proper mailing list. He was very lucky to find that Chris Ellison was prepared to help. After some initial tests, the production of the class finally started in July 2005, and it was mostly completed during the summer holidays of 2005. This article will describe the build process and the choices made along the way.

Paper size and orientation

Before putting anything on the paper, we needed to be sure we were using the correct paper size and orientation. However, as the idea was to place all content in `minipage` environments and then use pstricks' `\rput` to position the environments on the paper, we really didn't need to worry about page dimensions and margins for the user. So, we removed all margins and placed the origin `(0,0)` in the lower-left corner of the paper and `(\slidewidth,\slideheight)` in the upper-right corner. This provided an easy way for designers to create scalable styles for use with multiple paper types such as letter paper, a4 paper and screen ratio paper (4/3).

But what are these lengths `\slidewidth` and `\slideheight`? They will be determined from the paper type and orientation specified by the user and will be set to `.5\paperwidth` and `.5\paperheight`. We then magnify the DVI by a factor of two to have easy access to large fonts with the regular files `size10.clo` etcetera. This creates a useable DVI file,[2] a useable PS file (after processing with dvips), and a useable PDF file (after processing with ps2pdf).

To help the user when compiling to PDF, powerdot uses the `papersize` special to tell dvips which paper should be used. This way, the user does not need to specify the paper type with the `-t` command line option. However, there is a problem with this special. Most dvips configurations used today have a special `A4size` paper which, when a4 paper dimension are found in the `papersize` special, does not write the PostScript `a4` command to the PostScript file. When processing this PostScript file using ps2pdf without command line parameters, the program will not find a particular paper type and will default to letter paper. To avoid this problem, powerdot explicitly writes the `a4` command to the PostScript file when a4 paper is requested.[3,4]

Designer interface

So far, we have set up the paper dimensions and made sure that the user can get a proper DVI, PS or PDF file without much trouble or knowing about command line parameters. Now we have to make sure that new slide styles can easily be developed. This will be a huge improvement over prosper's complicated and basically absent designer interface.

Remember, we started with the idea of putting content on the paper in minipage environments using \rput. This gives rise to a very simple but powerful designer interface where all properties of the main components (slide title, text box, etcetera) can be controlled by keys which are defined using xkeyval [2]. These keys can be used in the \pddefinetemplate command, which has another argument that creates the background of the slide (using, for instance, pstricks). A special key, called ifsetup, can be used to specify to which setups all following keys should apply. For instance,

```
ifsetup={landscape,a4paper}
```

tells powerdot that all following keys should be used in case the user requested landscape a4 paper. The following, however,

```
ifsetup=landscape
```

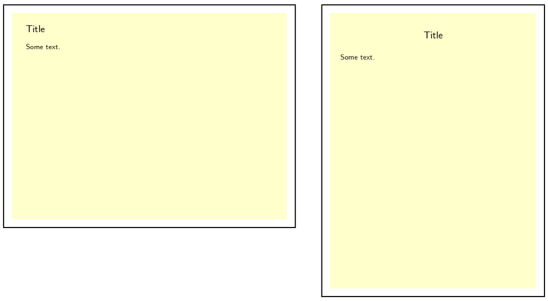
makes all following keys used in landscape orientation, but with any paper type. There is also a 'stand-alone' version of this key called \pdifsetup.

Finally, the \pddefinetemplate command allows us to use an existing template as basis for a new template, which further simplifies style development. Here is a simple example of the designer interface.

```
\documentclass[
  % orient=portrait
]{powerdot}
\pddefinetemplate{basic}{
  titlepos={.05\slidewidth,.91\slideheight},
  titlewidth=.9\slidewidth,
  textpos={.05\slidewidth,.85\slideheight},
  textwidth=.9\slidewidth,
  textfont=\raggedright\color{black}
}{%
  \psframe*[linecolor=yellow!20]%
  (0,0)(\slidewidth,\slideheight)%
}
\pddefinetemplate[basic]{slide}{%
  ifsetup=landscape,
  titlefont=\Large\raggedright\color{black},
  ifsetup=portrait,
  titlefont=\Large\centering\color{black}
}{}
\begin{document}
  \begin{slide}{Title}
    Some text.
  \end{slide}
\end{document}
```

The \pddefinetemplate command defines the slide template which will be based on the basic template. This template initializes the position of the main text box and the title and the text font to be used. In addition to the declarations coming from the basic template, the slide template specifies the title font. Note the use of the ifsetup key to choose different formatting for the slide title in landscape mode or portrait mode. In practice, this might considered to be inconsistent design, but here it just serves as an example. This example is very simple and the templates could easily be merged into one, but it clearly demonstrates the possibilities to reuse existing templates.

If we typeset the example above in both landscape and portrait orientation, we get the following output.



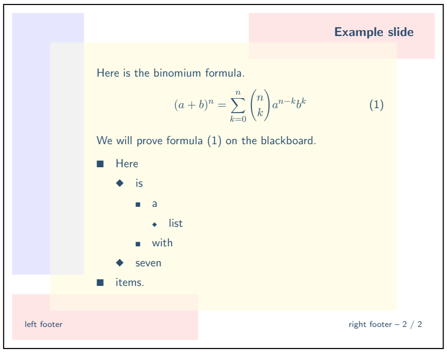
When a designer wants to do more fancy things which cannot be controlled by keys, powerdot supplies access to a variety of macros that do specific jobs and can be redefined to achieve these goals. Examples are \pd@title, which controls the typesetting of the presentation title, and \pd@slidetitle, which controls the typesetting of slide titles. By default, these macros just pass on their argument, but can be redefined to do arbitrary things.

As an example of the various possibilities of the design interface of powerdot, you can find samples of some of the currently available presentation styles in figures 1 to 4.

User interface

Most importantly, a new user interface needed to be developed which was both powerful and simple to use. Setting up the main characteristics of a presentation, like paper type, font size and style, is done via the \documentclass command. Other settings, like the footers, transition effects and layout of lists, is done via the \pdsetup command.

The user interface for making slides is kept very simple and is mainly formed by the slide environment.[5] This environment first stores the literal text of the body in a token register. This allows us to reuse the



Example slide

Here is the binomium formula.

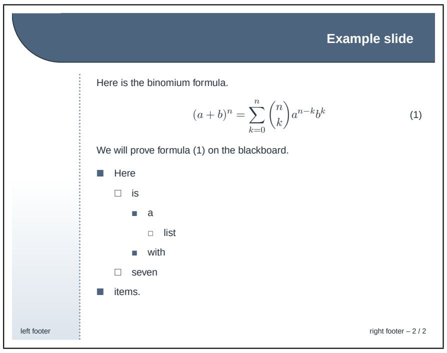
$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k \tag{1}$$

We will prove formula (1) on the blackboard.

- Here
  - is
    - a
      - list
    - with
  - seven
- items.

left footer right footer – 2 / 2

Figure 1. elcolors style



Example slide

Here is the binomium formula.

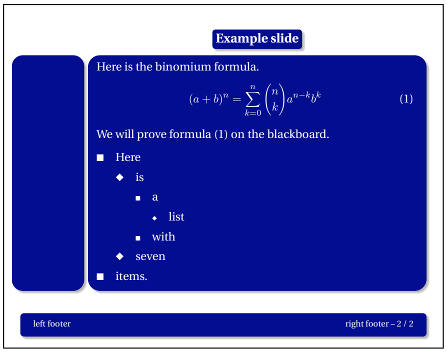
$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k \tag{1}$$

We will prove formula (1) on the blackboard.

- Here
  - is
    - a
      - list
    - with
  - seven
- items.

left footer right footer – 2 / 2

Figure 2. sailor style



Example slide

Here is the binomium formula.

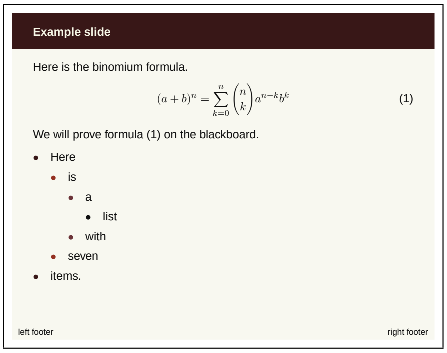
$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k \tag{1}$$

We will prove formula (1) on the blackboard.

- Here
  - is
    - a
      - list
    - with
  - seven
- items.

left footer right footer – 2 / 2

Figure 3. bframe style



Example slide

Here is the binomium formula.

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k \tag{1}$$

We will prove formula (1) on the blackboard.

- Here
  - is
    - a
      - list
    - with
  - seven
- items.

left footer right footer

Figure 4. paintings style

body later on. We do this by searching the input stream for the next occurrence of the `\end` command. If this command has the proper argument, namely `slide`, then we have found the end of the slide and we can start processing the content. If not, we add the text found so far to the token register and continue the search.

Now that we have the body 'in our hands', we can typeset it once and see what happens. The user could actually have specified an overlay command like `\onslide` or `\pause` in the slide. During the first run, these commands are executed and these are used to determine the remaining number of times that we need to typeset the body. This process creates several overlays using just one slide environment. Here is a simple example.[6]

```
\begin{slide}{My first slide}
  Hello \pause world!
\end{slide}
\begin{slide}{My second slide}
  \onslide{1-}{Hello} \onslide{2}{world!}
\end{slide}
```

This example creates two overlays for each slide. `Hello` will appear on both overlays for each slide, while `world!` appears only on every second overlay.

There is a drawback to using the technique described above to get the body of the environment and that is that category codes will be fixed in the text once we start typesetting it for the first time. Hence, constructions that rely on changing catcodes internally, like the `verbatim` environment, do not work inside the slide environment. This problem is easily worked around by storing the verbatim text outside the slide environment in a box and using that box inside the slide.

Supporting LaTeX commands

Of course, making a presentation is rather different from writing the article itself and by introducing new features, such as overlays, we might bring standard LaTeX constructions in trouble. Take for instance LaTeX counters. When repeatedly typesetting the same text, counter increases in that text (for instance by the equation environment) also get executed several times by the same text. This could lead to the same equation having different numbers on different overlays. This is easily overcome, however. We just record the value of some counters before typesetting the first overlay and reset it at the start of the next overlay. powerdot does this automatically for the counters `equation`, `figure`, and `table`. The user can easily add more counters to the list by using the `counters` key in the `\pdsetup` command.

Another example is the `\label` command. If the standard `\label` command would be executed on overlays, the user would always get `Multiply defined labels` errors.

prosper tried to solve this issue by executing `\labels` only on the first overlay. The error is obvious. Another idea would be to tell the user to always use `\label` inside an appropriate `\onslide` command with a single overlay specification. That, however, requires extra work from the user.

powerdot executes the `\label` only on the first overlay *where it is actually used*. This could, in fact, be overlay 37. The way it does this is by adding all labels defined on a slide to a list. If the list already includes the current label, this label is not executed again. The list is emptied at the start of every slide. The side effect of this system is that multiply-defined labels on the same slide cannot be detected anymore. However, multiply-defined labels on different slides still result in a warning in the log file of the user. This side effect is not considered very serious as the source of a slide is often rather short.

LyX support

To support the use of LyX [4] for creating powerdot presentations, the user interface should be able to deal with the restrictions set by LyX. One of the difficulties with LyX's interface is that it doesn't allow environments to have arguments. Instead, we have to use commands to indicate the beginning and end of a slide. When a powerdot LyX presentation is exported to LaTeX it looks like

```
\documentclass{powerdot}
\begin{document}
\lyxend\lyxslide{My first slide}
  Hello \pause world!
\lyxend\lyxslide{My second slide}
  \onslide{1-}{Hello} \onslide{2}{World}
\lyxend
\end{document}
```

Here, `\lyxend` is a harmless macro that is only used by `\lyxslide` as a delimiter. This interface can easily be extended by using the `\pddefinelyxtemplate` command in case a style defines custom templates. This command defines a control sequence that uses the underlying templates, like `\lyxslide` uses the `slide` template.

Hiding material

How do `\onslide` and `\pause` actually work when hiding material?[7] This is done using overlays offered by pstricks. We can use this system in the following way. On every slide, we initialize PostScript overlay 0. On that overlay, text will be visible. PostScript overlay 1 is used to make material invisible. This means that it will be typeset as usual by LaTeX, but that the material will not be visual in the output. Hence, the cursor will still be moved by the material. By switching to overlay 1 and back at the right times, we can hide any material we want. By switching to overlay 1 and not back anymore, we can hide all following material.

If we consider the example again and ignore all second (powerdot) overlays (as all material will be visible there), it comes basically down to executing the following.

```
\documentclass{powerdot}
\begin{document}
\makeatletter
\begin{slide}{My first slide}
  Hello \pst@Verb{(1) BOL} world!
\end{slide}
\begin{slide}{My second slide}
  Hello \pst@Verb{(1) BOL}world!\pst@Verb{(0) BOL}
\end{slide}
\makeatother
\end{document}
```

The `\pst@Verb` commands enter the switches to PostScript overlay 0 and 1 into the PostScript document via `\special`'s. We see that `\pause` will not return to overlay 0 afterwards, whereas `\onslide` does so. Hence, any following material would be invisible on powerdot overlay 2 on the first slide and not on the second.

Final details

The final task for the user interface was to fill in the details. An interface was necessary to create sections, table of contents entries, prevent `figure` and `table` environments from floating, create personal notes and handouts, and much more. Please have a look at the user documentation if you are interested in learning more about the powerdot class. The result of this holiday effort is a class that can create good-looking slides with a minimal amount of input from the designer and user, both when typing the source and when compiling it.

The future

The development of powerdot will of course continue and we have the plan to add new and innovative features to the class that will be very useful. When writing this, we are also working hard on, amongst many other things, multiple palettes per style (meaning the same design but different colors), a digital clock on slides and random elements (like dots and rings) to make presentations a little livelier. When you read this, powerdot might already have had an update in which all these features, and more, are present.

References

[1] Hendri Adriaens. HA-prosper package. `CTAN:/macros/latex/contrib/HA-prosper`.

[2] Hendri Adriaens. xkeyval package. `CTAN:/macros/latex/contrib/xkeyval`.

[3] Hendri Adriaens and Christopher Ellison. powerdot class. `CTAN:/macros/latex/contrib/powerdot`.

[4] LyX crew. LyX website. `http://www.lyx.org`.

[5] Frédéric Goualard and Peter Møller Neergaard. prosper class. `CTAN:/macros/latex/contrib/prosper`.

[6] Herbert Voß. PSTricks website. `http://pstricks.tug.org`.

[7] Timothy Van Zandt et al. PSTricks package, v1.07, 2005/05/06. `CTAN:/graphics/pstricks`.

Notes

1. At first, the name TEXciting was chosen, but that was abandoned due to associations with 'citations'.
2. For DVI viewers that understand PostScript `\specials`.
3. And the `letter` command for letter paper.
4. There is the `nopsheader` option which avoids writing the `papersize` special and the `a4` command. This should be used when the user can't use dvips without command line parameters, for instance, because the editor always inserts either `-tletter` or `-ta4`.
5. Most styles supply more templates, like the `wideslide` environment, but these work internally the same as the `slide` environment.
6. Please refer to the documentation for syntax details.
7. There are also versions of these macros that eat material or color it with another color than the text color.

Hendri Adriaens
hendri[at]uvt.nl

Chris Ellison
chris.ellison[at]gmail.com

Managing a network TeX installation under Windows

Keywords
MikTeX, TeXnicCenter, filename database, Windows, registry, roaming profiles, graphic file formats

This paper is about the MikTeX installation that I maintain for the Economics Department of the Rijksuniversiteit Groningen. We have long been the home of 4TeX. But when development of this project stopped, the time came that this TEX installation had to be replaced by something else. This something else was going to be MikTeX with TeXnicCenter as editor and front end.

There are various Windows editors which support MikTeX, *i.e.* editors which have menu items and buttons for compiling and viewing your TeX documents with MikTeX. Configuring *e.g.* TeXnicCenter or Winedt for MikTeX is almost automatic. TeXnicCenter is free, both in the beer and in the speech sense. The MikTeX site lists a few more free editors. LaTeXEditor (`http://www.ntu.edu.sg/home5/pg03053527/latexeditor/`) and Texmaker (`http://www.xm1math.net/texmaker/`) seem to have a similar focus as TeXnicCenter.

MikTeX itself comes with a configuration program, MikTeX Options or mo.exe[1], which has to be started from outside the editor.

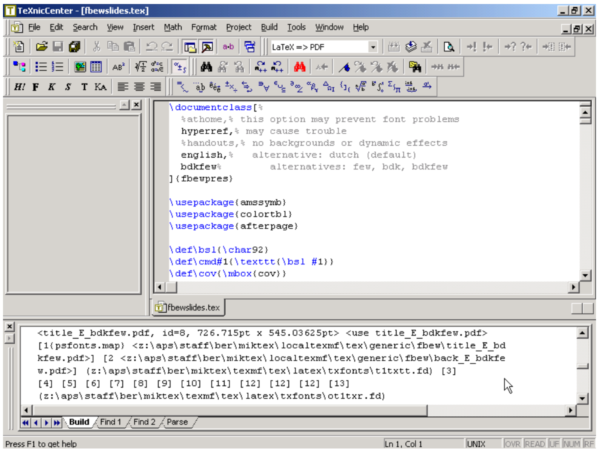
Over time, the MikTeX installation has been accumulating some add-ons, especially for handling graphics; see further on.

Moving from 4TeX to MikTeX

I didn't try to create a unified 4TeX-style interface for everything, and also didn't try to replicate the functionality of 4TeX, but I did collect the local macros, fonts and graphics from 4TeX which were still in use and put them into the MikTeX installation, sometimes with some minor tweaks.

I dropped support for the old LaTeX 2.09 since it would have meant real work for something that might not even get used.

The MikTeX installation was put online early in 2003. For a year and a half, MikTeX and 4TeX were available side by side, but in the end, after six months notice, I removed 4TeX from the network.


TeXnicCenter - [fbewslides.tex]
File Edit Search View Insert Math Format Project Build Tools Window Help
LaTeX => PDF
fbewslides.tex
Build
Find 1
Find 2
Parse
Press F1 to get help
Ln 1, Col 1
UNIX
OVR
READ
UF
NUM
RF

```
\documentclass[%
  %athome,% this option may prevent font problems
  hyperref,% may cause trouble
  %handouts,% no backgrounds or dynamic effects
  english,%   alternative: dutch (default)
  bdkfew%        alternatives: few, bdk, bdkfew
]{fbewpres}

\usepackage{amssymb}
\usepackage{colortbl}
\usepackage{afterpage}

\def\bsl{\char92}
\def\cmd#1{\texttt{\bsl #1}}
\def\cov{\mbox{cov}}
```

```
<title_E_bdkfew.pdf, id=8, 726.715pt x 545.03625pt> <use title_E_bdkfew.pdf>
[1{psfonts.map} <z:\aps\staff\ber\miktex\localtexmf\tex\generic\fbew\title_E_bd
kfew.pdf>] [2 <z:\aps\staff\ber\miktex\localtexmf\tex\generic\fbew\back_E_bdkfe
w.pdf>] (z:\aps\staff\ber\miktex\texmf\tex\latex\txfonts\t1txtt.fd) [3]
[4] [5] [6] [7] [8] [9] [10] [11] [12] [12] [12] [13]
(z:\aps\staff\ber\miktex\texmf\tex\latex\txfonts\ot1txr.fd)
```

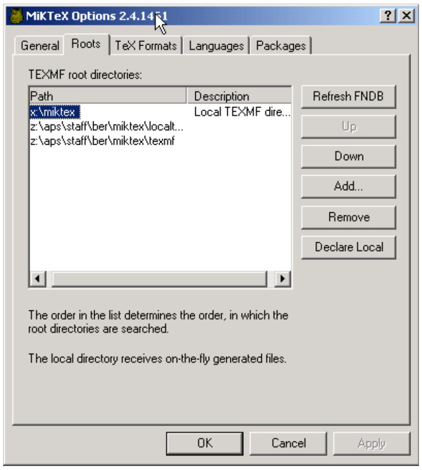
Figure 1. TeXnicCenter, a MikTeX frontend

Layout and contents of the installation

Texmf trees. As to the organization of macros, fonts and other support files, MikTeX is rather similar to other modern free TeX implementations: it organizes its files into *texmf trees*, which have a standardized structure: *e.g.* font-related files are in subdirectories fonts\tfm, fonts\type1 etc., and LaTeX macros are in tex\latex. Each tree follows such a structure and has its own filename database. Users can configure in which order the trees are going to be searched.

I configured the following three trees: a main tree for files coming from the MikTeX distribution; a department tree for local additions, including the files inherited from the 4TeX installation; and a user tree for people's own macros and other files. Users have write access only to their own user tree. With this setup, a MikTeX upgrade won't interfere with the department tree, and anything done to the network installation leaves user files alone.

Package selection. MikTeX is distributed as a setup program and a set of packages. The MikTeX Setup Wizard lets you choose between three initial package sets: small, large or everything, which you can modify later on. I picked the 'large' package set, and added and removed some packages afterwards.



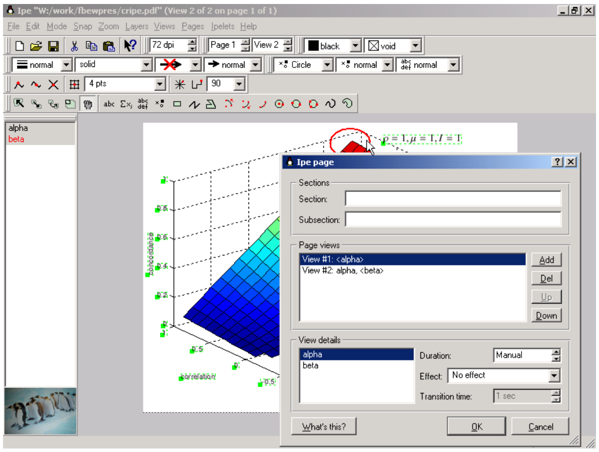
TEXMF root directories:

Path	Description
x:\miktex	Local TEXMF dire...
z:\aps\staff\ber\miktex\localt...	
z:\aps\staff\ber\miktex\texmf	

The order in the list determines the order, in which the root directories are searched.

The local directory receives on-the-fly generated files.

Figure 2. Defining trees and their priorities with MikTeX Options



Sections

Section:

Subsection:

Page views

View #1: <alpha>
View #2: alpha, <beta>

View details

alpha
beta

Duration: Manual

Effect: No effect

Transition time: 1 sec

Figure 3. The Ipe draw program has views or pages which are really assemblies of 'layers'.

Add-ons. Non-MikTeX add-ons originally included Ghostscript and GSview, but for the current edition, this was no longer necessary, since Ghostscript and GSview were already installed separately.

For better scripting, I included the Perl .exe and .dll files, but placed them outside the search path. If people have a need for Perl then they can install their own copy, without these two files getting in the way.

Graphics support

Draw programs. Our MikTeX installation includes a couple of draw programs. One of these is Ipe (`http://ipe.compgeom.org`), which has a few interesting features:

- □ It uses pdflatex in the background for typesetting text elements. You can tune typographic details with LaTeX preamble commands.
- □ It can import arbitrary pdf via a separate conversion utility.
- □ A drawing can be layered in the sense that it can be displayed incrementally in a pdf presentation. In fact, Ipe also advertises itself as a tool for making presentations.

A second draw program is LaTeXCAD, which generates LaTeX picture environments. It is very old and basic and is only included for people who still have old LaTeXCAD drawings in use.

Converters. I also added some PostScript-, .eps and .pdf conversion scripts, with desktop shortcuts which can be used as drag-and-drop targets. For conversion from pdf to EPS or Encapsulated PostScript I added the xpdf utilities.

I plan to write a basic GUI tool, custom tailored for our installation, which offers all available conversions from a single interface. Of course, the real work will be done by Ghostscript and other trusty command-line standbys.

There is also an installer for wmf2eps. This program offers a fairly practical way to make graphics from MS Office and other Windows programs available to LaTeX. It seems that not much has happened with it lately, but it still works well enough. It relies on a virtual printer driver, and therefore isn't a good candidate for a network install.

Its main advantage over simply printing to an eps file is that it calculates a tight boundingbox, rather than just turning an entire page into a graphic.

Recent versions of Ghostscript have a 'boundingbox output device'. There are now various scripts which use Ghostscript to fix boundingboxes. The as yet nonexistent GUI tool mentioned above should also offer such an option, as an alternative to using wmf2eps[2].

An installer

Installation of the network version involves:

- □ storing configuration information in the registry
- □ creating the user tree if it doesn't exist
- □ creating desktop and start menu shortcuts
- □ generating the filename database

Because our desktop systems are very standardized, none of this requires user input.

The filename database. MikTeX's texmf trees have an important and annoying difference with the more unixy varieties: the filename database of a tree is not stored with that tree, but in a designated 'local' tree which receives generated files. This local tree can only be the user tree. That means that it is up to the user to update his filename database if items are added to or moved around in the global installation.

The filename database can be generated from the MikTeX Options program but, fortunately for writers of installation scripts, it can also be done from the command line:

```
initexmf --update-fndb
```

It happens often enough that the installation fails because the user becomes impatient with the generation of the filename database. Without the help of this database, MikTeX becomes very very slow. To minimize the problem, I saved filename database generation till last in the installation process and preceded it with dire warnings about not interrupting the installer. If these warnings are ignored, then the filename database can still be generated after the fact from the MikTeX Options program. An alternative would be to copy a pregenerated filename database to the right place during installation.

Dealing with the registry

Under Windows, almost any configuration information is stored in the registry.

The registry contains a set of hierarchically organized keys. There are several root keys. The most important ones are `HKEY_CURRENT_USER` and `HKEY_LOCAL_MACHINE`, `HKCU` and `HKLM` in short. The `HKCU` part of the registry may be on a network drive. `HKLM` is normally in a subdirectory of the Windows directory.

The actual information is contained in *values* under those keys. Values are name–data pairs.

For users, there are very few reasons to edit the registry directly. There are almost always specialized menu entries and dialogs available, such as Tools menus and Control Panel entries.

Registry tools. If you do need to view or manipulate the registry directly, Windows contains a number of tools: you can browse the registry with regedit[3]. You can export and import registry keys to and from .reg files with either regedit or the command-line tool reg.exe. These .reg files are editable text files. Reg.exe may not be installed by default, though. Type '`reg /?`' for help.

Various programming languages, including Perl, VBScript and installer programs, also have functions for accessing the registry.

Finding out what registry values are needed. There are a number of techniques for identifying the registry settings that you need: one way would be to inspect the source code of the original installer.

A second method is browsing and searching the registry for likely strings, and testing afterwards whether you have captured enough to make the program work as intended. However, such testing can be time-consuming since you sometimes have to re-login or reboot before the changes take effect.

A third method is to export the registry to a text file before installation and after installation or first use, and compare the differences. There exist automated installers which do just this, but the GNU diff program works just fine.

You still have to decide what are the differences that matter. There will probably a lot of spurious differences. For example, most programs record windows positions and their most recently used files in the registry.

Also, some information which occurs only once can appear to occur multiple times. In particular, under Windows 2000 and later, the keys under HKCR are copies from keys under `HKLM\software\classes` and `HKCU\software\classes`.

All users or not: HKLM and KHCU. Often, when you install software, there is a choice whether or not to install for all users. If you do, keys are added under `HKLM\software`; otherwise under `HKCU\software`.

Where to look. The most important settings are under `software\<program name>` and under `software\classes` (either from `HKLM` or from `HKCU`). The keys under `classes` define file types and define what happens when you double-click a file in Windows Explorer. Command-line programs may not need any entries here.

Uninstall information can be found under `HKLM\software\Microsoft\Windows\CurrentVersion\Uninstall`. Our installer doesn't yet include an uninstaller.

Registry entries for MikTeX itself. MikTeX makes very modest use of the registry. It just records the locations of the texmf trees, stores uninstall information, and defines the .dvi filetype, associating it with the yap previewer.

I also added the MikTeX binaries directory to the search path, for those people who prefer to run MikTeX from the command line. On Windows 2000 and Windows XP the search path and other environment variables are stored in the registry; for the current user under `HKCU\software\environment`, for the local system under `HKLM\system\currentcontrolset\control\session manager\environment`.

Registry entries for Ghostscript and GSview. Ghostscript needs to record the location of the main dll and of its own fonts and support files. GSview defines the .eps and .ps file types and associates them with itself.

Registry entries for TeXnicCenter. TeXnicCenter stores a lot of information in the registry, but it can configure itself when it is started for the first time if it can find the MikTeX, Acrobat, Ghostscript and GSview registry settings. All the user has to do is to answer 'yes' a few times. I decided to leave configuration of TeXnicCenter to itself.

It is possible to rerun the TeXnicCenter configuration wizard at a later date. This may come in handy whenever MikTeX or Ghostscript or GSview has moved, or the Adobe Reader has been upgraded.

It would have been nice if TeXnicCenter checked at startup whether these programs are still at their previous location.

More installers

I started out with one installer, but now there are several.

A network installation for a LaTeX course. A second network installation was needed for a computer course for econometrics students. This installation is a slightly stripped-down version of the first one: no department tree, and without some of the add-ons.

A cd installation. Earlier, I had already made a cd containing the standard installers for MikTeX, Ghostscript, GSview and TeXnicCenter, and a copy of the department tree, plus a file with instructions how to put everything together.

There were two problems with this: it was complicated enough that some people preferred to let me install MikTeX for them, and other people figured that they might as well download and install MikTeX directly from the internet. Which was not exactly wrong, but differences in their setup sometimes made it difficult to debug their problems.

So I hope that the new installer cd has persuaded some people not to do a do-it-yourself install.

The differences. I already listed some of the differences between staff and student installations.

Some differences between network and cd installers are:

□ With the cd installer, users can choose locations for the main installation and for their own data. These locations are fixed in the network version.

□ The cd installer has to copy everything to the harddisk, whereas in the network version everything is already in place. In fact, re-running the network installer is no big deal. Which is just as well, since time and again configurations get messed up by a malfunctioning network or other mishaps.

□ The cd installer tests for Ghostscript and GSview. If they are missing, the user first has to install these programs, *e.g.* with the installers provided on the cd. The network version simply knows that Ghostscript and GSview are present and where they are.

□ The cd does an 'All users' install; the network version doesn't. Since on our network the `HKCU` part of the registry and the start menu are on the user's network drive, you can run MikTeX from any workstation on the network.

□ The cd only contains MikTeX fonts; not the additional department fonts. Adding fonts in MikTeX 2.4 is tricky at best. Adding them to systems that I didn't control caused too much grief.

It was not difficult to create the installer script as one main script with four different wrapper scripts.

I kept the installation and the installer on a Linux Samba server. I managed to put all 'real' files in a single directory tree, and to access these files through four different sets of symlinks. This prevented worries about keeping the versions in sync.

Installer programs. The standard way to distribute applications at the university is to create entries in NAL, or Novell Application Launcher, using Novell ZENworks. As I understand, ZEN identifies file system and registry differences before and after installation. With ZEN, an installer can make system changes which the user wouldn't have permissions for without ZEN. However, a first attempt to create such a NAL entry for MikTeX, done together with our NAL specialist, wasn't exactly plain sailing.

I needed something that I could develop and test on my own system. This was even more important for the student install for the LaTeX course, where I had to do everything through intermediaries who weren't even in the same building.

For the first edition, which didn't include a cd counterpart, I could make do with a batchfile with some embedded Perl code[4] for manipulating the search path.

The cd version of the second edition required user interaction, first for telling users to install Ghostscript if it wasn't found, second for asking users where MikTeX should be installed. So it was really time to switch to a GUI installer.

I picked NSIS (`http://nsis.sourceforge.net/`). It is completely scriptable and can be used from the command line[5]. It has functions for reading and writing the registry and for creating shortcuts. It offers string handling and conditionals. You can choose to what extent you want to package files into the installer itself, *i.e.* you can also tell the installer to copy files straight from the installation media to the target system.

The principal drawback of NSIS is very low-level string handling, which is quite painful if you are used to Perl string handling and regular expressions.

I have also heard good things about InnoSetup (`http://www.jrsoftware.org/isinfo.php`), but by then I was almost finished with my NSIS installer.

Development and testing

Virtual machines. Nowadays, you don't need physical test machines anymore; with software such as VMware you can create virtual guest machines for testing inside a host, *i.e.* inside your everyday computer. The hard disk of this guest computer is a very large file on the host's disk. Its screen can take up the entire physical screen, but it may also run inside a window, whatever is convenient. If host and guest have similar processors, performance can be quite decent.

VMware has versions for Windows and for Linux hosts. There are other options for virtual machines, both commercial and free: the Xen project (`http://www.xensource.com`) is getting a lot of attention, and may in time become a very interesting alternative. See also QEMU (`www.qemu.org`), Win4Lin (`http://win4lin.com`) and Virtual PC (`http://www.microsoft.com`[6]. Some of these emulators are focused more on running Windows applications than on creating a realistic test environment.

For testing installers, you can create a 'clean' virtual machine with just Windows and some indispensable programs installed. Then you can run simulations on copies of this virtual machine. Getting another clean test machine is just a matter of making a fresh copy of the original machine, which takes only minutes.

Virtual networking. For networking, I let VMware create what it calls a host-only network, with no direct access to an external network. This saved me the hassle of protecting virtual Windows machines against malware from the internet. I configured the Linux host as a Samba server, with the MikTeX installation in a Samba share, and the user's home directory in another share.

Roaming profiles. The university started using *roaming profiles*. The idea is to place user configuration data as much as possible on their own network home drive. These user configuration data include *e.g.* their start menu and the HKCU part of the registry.

With Samba, roaming profiles means configuring the (or a) Samba server as a PDC or Primary Domain Controller. This is no fun. It means creating things called machine accounts for the client machines, and painstakingly reading Samba documentation. A very helpful and funny guide was 'The Unoffical Samba HOWTO'. You can find this document via the Samba site. Its current location is `http://hr.uoregon.edu/davidrl/docs/samba.html`.

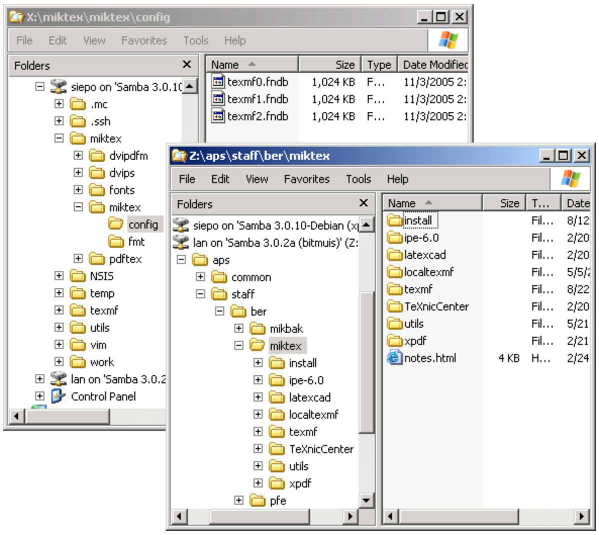


Figure 4. The Samba shares seen from the Windows client machine. It doesn't make a difference whether the Windows machine is physical or virtual.

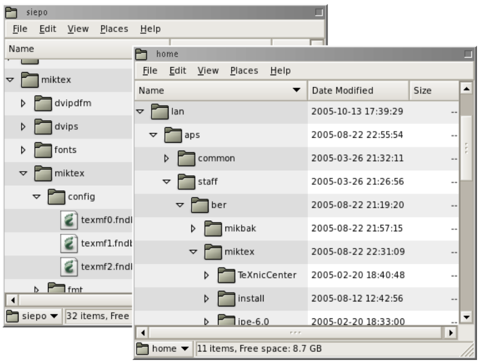

siepo
File Edit View Places Help
Name
miktex
dvipdfm
dvips
fonts
miktex
config
texmf0.fndl
texmf1.fndl
texmf2.fndl
fmt
siepo
32 items, Free
home
File Edit View Places Help
Name Date Modified Size
lan 2005-10-13 17:39:29 --
aps 2005-08-22 22:55:54 --
common 2005-03-26 21:32:11 --
staff 2005-03-26 21:26:56 --
ber 2005-08-22 21:19:20 --
mikbak 2005-08-22 21:57:15 --
miktex 2005-08-22 22:31:09 --
TeXnicCenter 2005-02-20 18:40:48 --
install 2005-08-12 12:42:56 --
ipe-6.0 2005-02-20 18:33:00 --
home
11 items, Free space: 8.7 GB

Figure 5. The Samba shares seen from the Linux server. It makes no difference whether the server is a separate machine, a VMware host or a second VMware guest machine.

For testing, I now make clean MikTeX-free profiles, with just the worst default Windows settings fixed, and work with copies of those clean profiles, just as I already did with guest machines.

Disappearing file types. In theory, with MikTeX and TeXnicCenter, roaming profiles should work perfectly: there should be no need to install or configure anything on the machine itself. In practice, definitions of filetypes under HKCU, *i.e.* all keys and values under `HKCU\software\classes`, got lost in between logins – in real life, not in my test setup. A good workaround is to recreate those filetype definitions automatically at every login, *e.g.* via the Start, Programs, startup menu.

I made some mistakes here: I could have caught the problem beforehand if I had been more rigorous with using clean test machines, and my original workaround was a good deal clumsier than the fix with the startup menu.

Final thoughts

I would have liked to add some nice generalities, but all I can think of is that Windows is still evil and has been giving me a hard time.

Notes

1. MikTeX also has a package manager. But of course that is not useful to users who don't have write access to the installation.
2. There is also a Linux program called wmf2eps, but I have had mixed results with it. It seems better to convert wmf and emf graphics to .eps or .pdf on the original system *before* trying to use them elsewhere.
3. Under earlier versions of Windows, regedit and regedt32 each could do things that the other couldn't. Under Windows XP, regedit combines the functionality of the earlier regedit and regedt32. Its version of regedt32 simply starts up regedit.
4. That is, the batchfile calls Perl with the `-x` switch and itself as parameter; see the perlrun manpage.
5. For GUI addicts, there is an interface with some buttons to push. There are also third-party editors with a GUI for building dialog boxes.
6. Virtual PC was bought from Connectix by Microsoft in the second half of 2003. The Macintosh version of Virtual PC was at the time the only real option for running Windows on the Mac. I, along with many other Mac owners, was duly shocked by this sell-out. But in the meantime, other emulators appeared, and now that the Mac is moving to Intel, we can hope for VMware-quality virtual machines on Mac OS X from other companies than Microsoft.

Siep Kroonenberg
`siepo@cybercomm.nl`

An Introduction to MetaUML

Exquisite UML Diagrams in MetaPost

Abstract
MetaUML is a GNU GPL MetaPost library for typesetting exquisite UML (Unified Modeling Language) diagrams. MetaUML offers a highly customizable, object-oriented API, designed with the ease of use in mind. This paper presents usage examples as well as a description of MetaUML infrastructure. This infrastructure may prove useful for general MetaPost typesetting, providing object-oriented replacements and enhancements to functionalities offered by the boxes package.

Keywords
MetaPost, TeX, LaTeX, UML, class diagram, state machine diagram, use case diagram, activity diagram

Introduction

Figure 1 presents a gallery of diagrams created by MetaUML (Gheorghies (2005)).

The code which generates these diagrams is quite straightforward, combining a natural object-oriented parlance with the power of MetaPost equation solving; for more information on MetaPost see Hobby (1992).

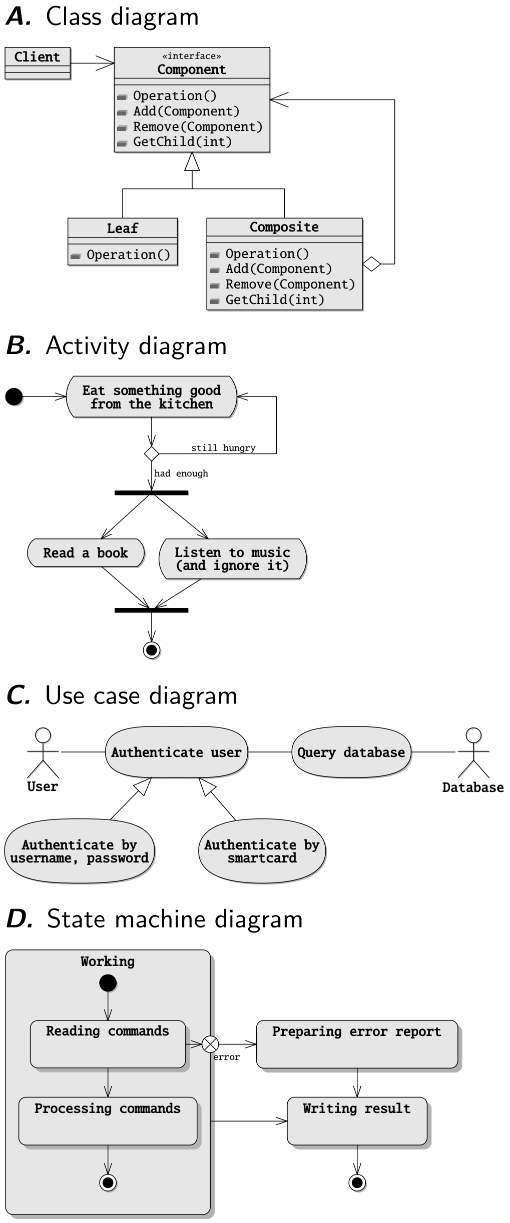
An UML class, for example, can be defined as follows:

```
Class.A("MyClass")
  ("attr1: int", "attr2: int")
  ("method1(): void", "method1(): void");
```

This piece of code creates an instance of `Class`, which will be afterward identified as `A`. This object has the following content properties: a name (`MyClass`), a list of attributes (`attr1`, `attr2`) and a list of methods (`method1`, `method2`). The one thing remaining before actually drawing `A` is to set its location:

```
A.nw = (0, 0);
drawObject(A);
```

In `A.nw` we refer to the "north-west" of the class rectangle, that is to its upper-left corner. In general, every MetaUML object has the positioning properties given in figure 2. These properties are used to set where to draw a given object, whether by assigning them absolute values, or by setting them relatively



A. Class diagram

B. Activity diagram

C. Use case diagram

D. State machine diagram

Figure 1. UML diagrams created by MetaUML.

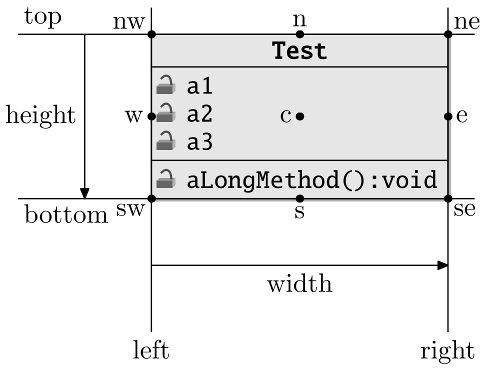


Figure 2. Positioning properties of any MetaUML object (here a class object is depicted).

to other objects. Suppose that we have defined two classes A and B. Then the following code would give a conceivable positioning:

```
A.nw = (0,0);
B.e = A.w + (-20, 0);
```

After the objects are drawn, one may draw links between them, such as inheritance or association relations between classes in class diagrams, or transitions between states in state machine diagrams. Whichever the purpose is, MetaUML provides a generic way of drawing an edge in a diagram's graph:

```
link(how-to-draw-information)(path-to-draw);
```

The "how to draw information" is actually an object which defines the style of the line (e.g. solid, dashed) and the appearance of the heads (e.g. nothing, arrow, diamond). One such object, called `inheritance`, defines a solid path ending in a white triangle. The `path-to-draw` parameter is simply a MetaPost path. For example, the following code can be used used to represent that class B is derived from A:

```
link(inheritance)(B.e -- A.w);
```

Note that the direction of the path is important, and MetaUML uses it to determine the type of adornment to attach at the link ends (if applicable). In our example, a white triangle, denoting inheritance, points towards the end of the path, that is towards class A.

To sum up, we present a short code and the resulting diagram (figure 3). This is typical for everything else in MetaUML. The positioning of A does not need to be explicitly set because "floating" objects are automatically positioned at (0,0) by their draw method.

```
input metauml;
beginfig(1);
  Class.A("A")()();
  Class.B("B")()();
  B.e = A.w + (-20, 0);
  drawObjects(A, B);
  link(inheritance)(B.e -- A.w);
endfig;
end
```

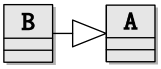


Figure 3. Example of MetaUML. Everything else works the same.

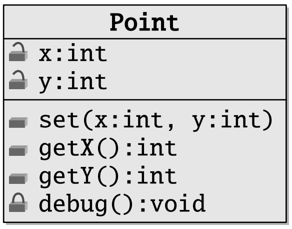


Figure 4. Class usage: name, attributes, methods and visibility markers.

From a user's perspective, this is all there is to MetaUML. With a reference describing how other UML elements are created, one can set out to typeset arbitrary complex diagrams.

Class Diagrams

A class is created as follows:

```
Class.name(class-name)
  (list-of-attributes)
  (list-of-methods);
```

The suffix `name` gives a name to the `Class` object (which, of course, represents an UML class). The name of the UML class is a string given by `class-name`; the attributes are given as a comma separated list of strings, `list-of-attributes`; the methods are given as a comma separated list of strings, `list-of-attributes`. The list of attributes and the list of methods may be void.

Each of the strings representing an attribute or a method may begin with a visibility marker: "+" for public, "#" for protected and "—" for private. MetaUML interprets this marker and renders a graphic stereotype in form of a lock which may be opened, semi-closed and closed, respectively.

The following code yields the diagram in figure 4.

```
Class.A("Point")
  ("#x:int", "#y:int")
  ("+set(x:int, y:int)",
   "+getX():int",
   "+getY():int",
   "-debug():void");
drawObject(A);
```

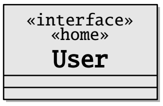


Figure 5. Class usage: stereotypes.

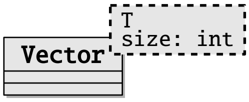


Figure 6. Class usage: templates.

Stereotypes

After a class is created, its stereotypes may be specified by using the macro `classStereotypes`:

```
classStereotypes.name(list-of-stereotypes);
```

Here, `name` is the object name of a previously created class and `list-of-stereotypes` is a comma separated list of strings. Here is an example along with the resulting diagram (figure 5).

```
Class.A("User")()();
classStereotypes.A("<<interface>>", "<<home>>");

drawObject(A);
```

Parametrized Classes (Templates)

The most convenient way of typesetting a class template in MetaUML is to use the macro `ClassTemplate`. This macro creates a visual object which is appropriately positioned near the class object it adorns.

```
ClassTemplate.name(list-of-templates)
                (class-object);
```

The `name` is the name of the template object, `list-of-templates` is a comma separated list of strings and the `class-object` is the name of a class object.

The code below results in the diagram from figure 6.

```
Class.A("Vector")()();
ClassTemplate.T("T", "size: int")(A);

drawObjects(A, T);
```

The macro `Template` can also be used to create a template object, but this time the resulting object can be positioned freely.

```
Template.name(list-of-templates);
```

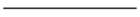
Of course, one can specify both stereotypes and template parameters for a given class.

Types of Links

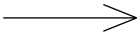
In this section we enumerate the relations that can be drawn between classes by means of MetaUML macros. Suppose that we have the declared two points, `A` (on the left) and `B` (on the right):

```
pair A, B;
A = (0,0);
B = (50,0);
```

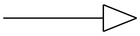
```
Bidirectional association.
link(association)( A -- B );
```



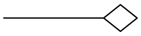
```
Unidirectional association.
link(associationUni)( A -- B );
```



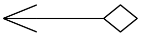
```
Inheritance. link(inheritance)( A -- B );
```



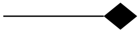
```
Aggregation. link(aggregation)( A -- B );
```



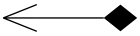
```
Unidirectional aggregation.
link(aggregationUni)( A -- B );
```



```
Composition. link(composition)( A -- B );
```



```
Unidirectional composition.
link(compositionUni)( A -- B );
```



Associations

In UML an association typically has two of association ends and may have a name specified for it. In turn, each association end may specify a multiplicity, a role, a visibility, an ordering. These entities are treated in MetaUML as pictures having specific drawing information (spacings, font).

The first method of creating association "items" is by giving them explicit names. Having a name for an association item comes in handy when referring to its properties is later needed (see the non UML-compliant diagram in figure 7). Note that the last parameter of the macro `item` is an equation which uses the item name to perform positioning.

```
Class.P("Person")()();
Class.C("Company")()(); % drawing code ommited

item.aName(iAssoc)("works for")
         (aName.s = .5[P.w, C.w]);
draw aName.n -- (aName.n + (20,20));
label.urt("association name" infont "tyxtt",
         aName.n + (20,20));
```

However, giving names to every association item may become an annoying burden (especially when

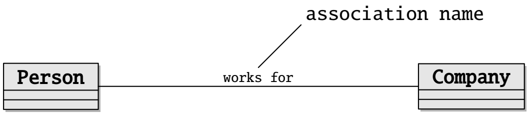


Figure 7. Referring to the properties of association items.

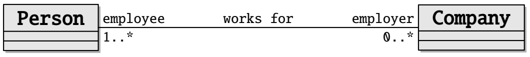


Figure 8. Anonymous association items.

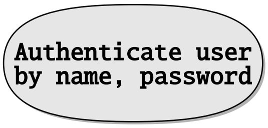


Figure 9. Usecase example.

there are many of them). Because of this, MetaUML also allows for “anonymous items”. In this case, the positioning is set by an equation which refers to the anonymous item as obj (figure 8).

```
% P and C defined as in the previous example

item(iAssoc)("employee")(obj.sw = P.e);
item(iAssoc)("1..*")(obj.nw = P.e);

% other items are drawn similarly
```

Use Case Diagrams

Use Cases

An use case is created by the macro Usecase:

```
Usecase.name(list-of-lines);
```

The list-of-lines is a comma separated list of strings. These strings are placed on top of each other, centered and surrounded by the appropriate visual UML notation.

Use case example (result in figure 9):

```
Usecase.U("Authenticate user",
          "by name, password");
drawObject(U);
```

Actors

An actor is created by the macro Actor:

```
Actor.name(list-of-lines);
```

Here, list-of-lines represents the actor’s name. For convenience, the name may be given as a list of strings which are placed on top of each other, to provide support for the situations when the role is quite long. Otherwise, giving a single string as an argument to the Actor constructor is perfectly fine.

Actor example (result in figure 10):

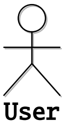


Figure 10. Actor example.

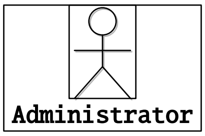


Figure 11. Actor example, accessing the “human”.

```
Actor.A("User");
drawObject(A);
```

Note that one may prefer to draw diagram relations positioned relatively to the visual representation of an actor (the “human”) rather than relatively to the whole actor object (which also includes the text). Because of that, MetaUML provides access to the “human” of every actor object actor by means of the sub-object actor.human. Figure 11 gives the result of the code below:

```
Actor.A("Administrator");
drawObject(A);
draw objectBox(A);
draw objectBox(A.human);
```

Note that in MetaUML objectBox(X) is equivalent to X.nw -- X.ne -- X.se -- X.sw -- cycle for every object X.

Types of Links

Some of the types of links defined for class diagrams (such as inheritance, association etc.) can be used with similar semantics within use case diagrams.

Activity Diagrams

Begin and End

The begin and the end of an activity diagram can be marked by using the macros Begin and End, respectively. The constructors of these visual objects take no parameters:

```
Begin.beginName;
End.endName;
```

Figure 12 gives the output of the code:

```
Begin.b;
End.e;
b.nw = (0,0);
e.nw = (20, 20);

drawObjects(b, e);
```

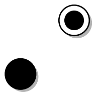


Figure 12. Begin and end in an activity diagram.

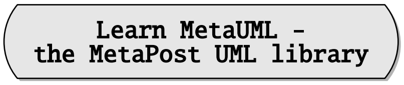


Figure 13. Activity example.

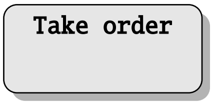


Figure 14. State example.

Activity

An activity is constructed as follows:

```
Activity.name(list-of-strings);
```

The parameter `list-of-strings` is a comma separated list of strings. These strings are centered on top of each other to allow for the accommodation of a longer activity description within a reasonable space.

An example is given in figure 13

```
Activity.A("Learn MetaUML -",
           "the MetaPost UML library");
drawObject(A);
```

Types of Links

In activity diagrams, transitions between activities are needed. They are typeset as in the example below. Figure 15 shows such a transition rendered. This type of link is also used for state machine diagrams.

```
link(transition)( pointA -- pointB );
```

State Diagrams

The constructor of a state allows for aggregated substates:

```
State.name(state-name)(substates-list);
```

The parameter `state-name` is a string or a list of comma separated strings representing the state's name or description. The `substates-list` parameter is used to specify the substates of this state as a comma separated list of objects; this list may be void.

Figure 14 presents a simple state, rendered by the following code:

```
State.s("Take order")();
drawObject(s);
```

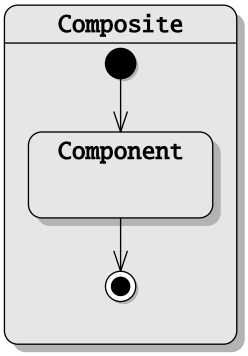


Figure 15. State example: composite states.

Composite States

A composite state is defined by enumerating at the end of its constructor the inner states. Interestingly enough, the composite state takes care of drawing the sub-states it contains. The transitions must be drawn after the composite state, as seen in the next example (figure 15):

```
Begin.b;
End.e;
State.c("Component")();
State.composite("Composite")(b, e, c);

b.midx = e.midx = c.midx;
c.top = b.bottom - 20;
e.top = c.bottom - 20;

composite.info.drawNameLine := 1;
drawObject(composite);

link(transition)(b.s -- c.n);
link(transition)(c.s -- e.n);
```

Internal Transitions

Internal transitions can be specified by using the macro:

```
stateTransitions.name(list-transitions);
```

Identifier `name` gives the state object whose internal transitions are being set, and parameter `list-transitions` is a comma separated string list. Figure 16 presents the result of the code below.

```
State.s("An interesting state",
        "which is worth mentioning")();
stateTransitions.s(
  "OnEntry / Open eyes",
  "OnExit  / Sleep well");
s.info.drawNameLine := 1;

drawObject(s);
```

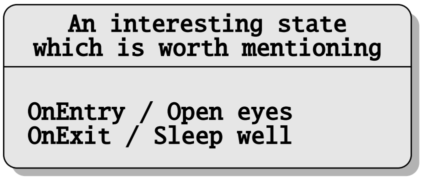


Figure 16. State example: internal transitions.

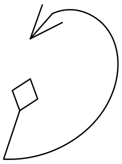


Figure 17. Link paths can be arbitrary complex in MetaUML: the heads are properly drawn.

Special States

Similarly to the usage of Begin and End macros, one can define history states, exit/entry point states and terminate pseudo-states, by using the following constructors.

```
History.nameA;
ExitPoint.nameB;
EntryPoint.nameC;
Terminate.nameD;
```

Drawing Paths

The link macro is powerful enough to draw relations following arbitrary paths (figure 17):

```
za = (10,10);
zb = (80,-10);
path cool;
cool := za .. za+(20,10) ..
        zb+(20,-40) ..
        zb+(-10,-30) -- zb;
link(aggregationUni)(cool);
```

Regardless of how amusing this feature might be, it does become a bit of a nuisance to use it in its bare form. When typesetting UML diagrams in good style, one generally uses rectangular paths. It is for this kind of style that MetaUML offers extensive support, providing a “syntactic sugar” for constructs which can otherwise be done by hand, but with some extra effort.

Manhattan Paths

The “Manhattan” path macros generate a path between two points consisting of one horizontal and one vertical segment. The macro pathManhattanX generates first a horizontal segment, while the macro pathManhattanY generates first a vertical segment. In MetaUML it also matters the direction of a path, so you can choose to reverse it by using rpathManhattanX and rpathManhattanY (note the prefix “r”):

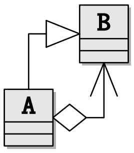


Figure 18. Manhattan paths.

```
pathManhattanX(A, B)
pathManhattanY(A, B)

rpathManhattanX(A, B)
rpathManhattanY(A, B)
```

Figure 18 shows these macros at work:

```
Class.A("A")()();
Class.B("B")()();

B.sw = A.ne + (10,10);
drawObjects(A, B);

link(aggregationUni)
   (rpathManhattanX(A.e, B.s));
link(inheritance)
   (pathManhattanY(A.n, B.w));
```

Stair Step Paths

These path macros generate stair-like paths between two points. The “stair” can “rise” first in the direction of Ox axis (pathStepX) or in the direction of Oy axis (pathStepY). How much should a step rise is given by an additional parameter, delta. Again, the macros prefixed with “r” reverse the direction of the path given by their unprefixed counterparts.

```
pathStepX(A, B, delta)
pathStepY(A, B, delta)

rpathStepX(A, B, delta)
rpathStepY(A, B, delta)
```

Figure 19 shows these macros at work:

```
stepX:=60;
link(aggregationUni)
   (pathStepX(A.e, B.e, stepX));

stepY:=20;
link(inheritance)
   (pathStepY(B.n, A.n, stepY));
```

Horizontal and Vertical Paths

There are times when drawing horizontal or vertical links is required, even when the objects are not properly aligned. To this aim, the following macros are use-

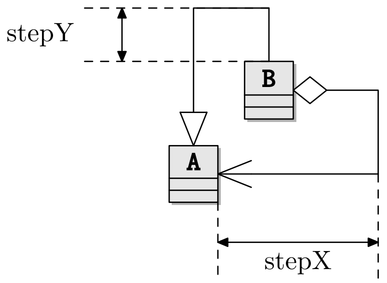


Figure 19. Stair step paths.

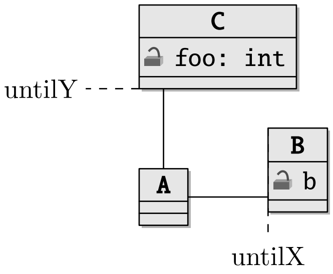


Figure 20. Horizontal and vertical paths.

ful:

```
pathHorizontal(pA, untilX)
pathVertical(pA, untilY)

rpathHorizontal(pA, untilX)
rpathVertical(pA, untilY)
```

A path created by `pathHorizonal` starts from the point `pA` and continues horizontally until coordinate `untilX` is reached. The macro `pathVertical` constructs the path dually, working vertically. The prefix “`r`” reverses the direction of the path.

Figure 20 gives an usage example:

```
untilX := B.left;
link(association)
   (pathHorizontal(A.e, untilX));

untilY:= C.bottom;
link(association)
   (pathVertical(A.n, untilY));
```

Direct Paths

A direct path can be created with `directPath`. The call `directPath(A, B)` is equivalent to `A -- B`.

Paths between Objects

Using the constructs presented above, it is clear that one can draw links between diagram objects, using a code like:

```
link(transition)(directPath(objA.nw, objB.se));
```

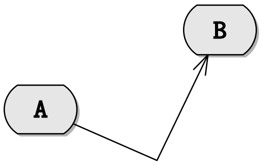


Figure 21. The `pathCut` macro at work.

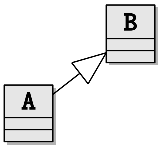


Figure 22. Direct linking between objects with `clink`.

There are times however this may yield unsatisfactory visual results, especially when the appearance of the object’s corners is round. MetaUML provides the macro `pathCut` whose aim is to limit a given path exactly to the region outside the actual borders of the objects it connects. The macro’s synopsis is:

```
pathCut(thePath)(objectA, objectB)
```

Here, `thePath` is a given MetaPost path and `objectA` and `objectB` are two MetaUML objects. By contract, each MetaUML object of type, say, `X` defines a macro `X_border` which returns the path that surrounds the object. Because of that, `pathCut` can make the appropriate modifications to `thePath`.

The following code demonstrates the benefits of the `pathCut` macro (figure 21):

```
z = A.se + (30, -10);
link(transition)
   (pathCut(A, B)(A.c--z--B.c));
```

Direct Paths between Centers. At times is quicker to just draw direct paths between the center of two objects, minding of course the object margins. The macro which does this is `clink`:

```
clink(how-to-draw-information)(objA, objB);
```

The parameter `how-to-draw-information` is the same as for the macro `link`; `objA` and `objB` are two MetaUML objects.

Figure 22 gives the output of the following code:

```
  clink(inheritance)(A, B);
```

The MetaUML Infrastructure

MetaPost is a macro language based on equation solving. Using it may seem quite tricky at first for a programmer accustomed to modern object-oriented languages. However, the great power of MetaPost consists in its versatility. Indeed, it is possible to write a

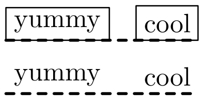


Figure 23. Motivation for not using boxes: the bottom alignment is imperfect.

system which mimics quite well object-oriented behavior. Along this line, METAOBJ (Roegel (2002)) is a library worth mentioning: it provides a high-level objects infrastructure along with a battery of predefined objects.

Surprisingly enough, MetaUML does not use METAOBJ. Instead it uses a custom written, lightweight object-oriented infrastructure, provisionally called “util”. The fact that METAOBJ’s source consists of a huge file which is rather hard to follow and understand contributed to this decision.

Another library that has some object-oriented traits is the boxes library, which comes with the standard MetaPost distribution. Early versions of MetaUML did use boxes as an infrastructure, but eventually it had to be abandoned. The main reason was that it was difficult to achieve good visual results when stacking texts (more on that further on). Also, it had a degree of flexibility which became apparent to be insufficient.

Motivation

Suppose that we want to typeset two texts with their bottom lines aligned, using boxit (figure 23):

```
boxit.a ("yummy");
boxit.b ("cool");

a.nw = (0,0); b.sw = a.se + (10,0);

drawboxed (a, b); % or drawunboxed(a,b)
draw a.sw -- b.se dashed evenly
   withpen pencircle scaled 1.1;
```

Note that “yummy” *looks* slightly higher than “cool”: this is unacceptable when, in an UML class diagram, roles are placed at the ends of a horizontal association. Regardless of default spacing being smaller in the util library, the very same unfortunate misalignment effect rears its ugly head (figure 24):

```
Picture.a("yummy");
Picture.b("cool");
% comment next line for unboxed
a.info.boxed := b.info.boxed := 1;

b.sw = a.se + (10,0);

drawObjects(a, b);
```

However, the strong point of util is that we have a recourse to this problem (figure 25):

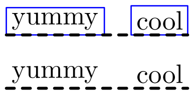


Figure 24. Misalignment occurs by default with the util library, but this can be configured not to happen.

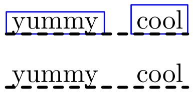


Figure 25. The util package provides good alignment.

```
iPict.ignoreNegativeBase := 1;

Picture.a("yummy");
Picture.b("cool");
% the rest the same as above
drawObjects(a, b);
```

The Picture Macro

We have seen previously the line iPict.ignoreNegativeBase := 1. Who is iPict and what is it doing in our program? MetaUML aims at separating the “business logic” (what to draw) from the “interface” (how to draw). In order to achieve this, it records the “how to draw” information within the so-called Info structures. The object iPict is an instance of PictureInfo structure, which has the following properties (or attributes):

```
left, right, top, bottom
ignoreNegativeBase
boxed, borderColor
```

The first four attributes specify how much space should be left around the actual item to be drawn. The marvelous effect of ignoreNegativeBase has just been shown (off) while the last two attributes control whether the border should be drawn (when boxed=1) and if drawn, in which color.

There’s one more thing: the font to typeset the text in. This is specified in a FontInfo structure which has two attributes: the font name and the font scale. This information is kept within the PictureInfo structure as a contained attribute iFont. Both FontInfo and PictureInfo have “copy constructors” which can be used to make copies. We have already the effect of these copy constructors at work, when we used:

```
Picture.a("yummy");
a.info.boxed := 1;
```

A copy of the default info for a picture, iPict, has been made within the object a and can be accessed as a.info. Having a copy of the info in each object may seem like an overkill, but it allows for a fine grained control of the drawing mode of each individual object.

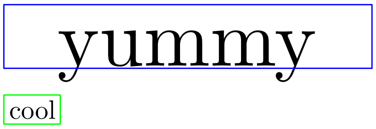


Figure 26. Having predefined configurations yields short, convenient code.

This feature comes in very handy when working with a large number of settings, as it is the case for MetaUML.

Let us imagine for a moment that we have two types of text to write: one with a small font and a small margin and one with a big font and a big margin. We could in theory configure each individual object or set back and forth global parameters, but this is far for convenient. It is preferable to have two sets of settings and specify them explicitly when they are needed. The following code could be placed somewhere in a configuration file and loaded before any beginfig macro:

```
PictureInfoCopy.iBig(iPict);
iBig.left := iBig.right := 20;
iBig.top := 10;
iBig.bottom := 1;
iBig.boxed := 1;
iBig.ignoreNegativeBase := 1;
iBig.iFont.name := defaultfont;
iBig.iFont.scale := 3;

PictureInfoCopy.iSmall(iPict);
iSmall.boxed := 1;
iSmall.borderColor := green;
```

Below is an usage example of these definitions (result in figure 26). Note the name of the macro: EPicture. The prefix comes form “explicit” and it's used to acknowledge that the “how to draw” information is set explicitly, as opposed to the Picture macro where the info member defaults to iPict.

```
EPicture.a(iBig)("yummy");
EPicture.b(iSmall)("cool");
% you can still modify a.info and b.info

b.sw = a.se + (10,0);

drawObjects(a, b);
```

Stacking Objects

It is possible to stack objects, much in the style of setboxjoin from boxes library (figure 27).

```
Picture.a0("yummy");
Picture.a1("cool");
Picture.a2("fool");

setObjectJoin(pa.sw = pb.nw);
```

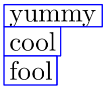


Figure 27. Stacking objects.

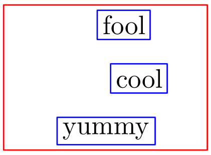


Figure 28. Grouping objects.

```
joinObjects(scantokens listArray(a)(3));
drawObjects(scantokens listArray(a)(3));
% or drawObjects (a0, a1, a2);
```

The listArray macro provides here a shortcut for writing a0, a1, a2. This macro is particularly useful for generic code which does not know beforehand the number of elements to be drawn. Having to write the scantokens keyword is admittedly a nuisance, but this is required.

The Group Macro

It is possible to group objects in MetaUML. This feature is the cornerstone of MetaUML, allowing for the easy development of complex objects, such as composite stats in state machine diagrams.

Similarly to the macro Picture, the structure GroupInfo is used for specifying group properties; its default instantiation is iGroup. Furthermore, the macro EGroup explicitely sets the layout information. Figure 28 results from the code below:

```
iGroup.left:=20;
iGroup.right:=15;
iGroup.boxed:=1;
iPicture.boxed:=1;

Picture.a("yummy");
Picture.b("cool");
Picture.c("fool");

b.nw = a.nw + (20,20);  % A
c.nw = a.nw + (15, 40); % B

Group.g(a, b, c);
g.nw = (10,10); % C

drawObject(g);
```

Note that after some objects are grouped, they can all be drawn by invoking the drawObject macro solely on the group that aggregates them. Another important remark is that it is necessary only to set the relative positioning of objects within a group (line A and

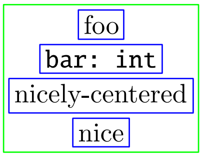


Figure 29. An example of a picture stack.

B); afterward, one can simply "move" the group to a given position (line C), and all the contained objects will move along.

The PictureStack Macro

The PictureStack macro is a syntactic sugar for a set of pictures, stacked according to predefined equations and grouped together (figure 29).

```
iStack.boxed := 1;
iStack.iPict.boxed := 1;
PictureStack.myStack("foo",
  "bar: int" infont "tyxtt",
  "nicely-centered" infont defaultfont,
  "nice")("vcenter");

drawObject(myStack);
```

Note the last parameter of the macro PictureStack, here vcenter. It is used to generate appropriate equations based on a descriptive name. The spacing between individual picture objects is set by the field iStack.spacing. Currently, the following alignment names are defined: vleft, vright, vcenter, vleftbase, vrightbase, vcenterbase. All these names refer to vertical alignment (the prefix "v"); alignment can be at left, right or centered. The variants having the suffix "base" align the pictures so that iStack.spacing refer to the distance between the bottom lines of the pictures. The unsuffixed variants use iStack.spacing as the distance between one's bottom line and the next's top line.

The "base" alignment is particularly useful for stacking text, since it offers better visual appearance when iPict.ignoreNegativeBase is set to 1.

Components Design

Each MetaUML component (e.g. Picture, PictureStack, Class) is designed according to an established pattern. This section gives more insight on this.

In order to draw a component, one must know the following information:

□ what to draw, or what are the elements of a component.
□ how to draw, or how are the elements positioned in relation to each other within the component
□ where to draw

For example, in order to draw a picture object we must know, respectively:

□ what is the text or the native picture that needs to be drawn
□ what are the margins that should be left around the contents
□ where is the picture to be drawn

Why do we bother with these questions? Why don't we just simply draw the picture component as soon as it was created and get it over with? That is, why doesn't the following code just work?

```
Picture.pict("foo");
```

Well, although we have the answer to question 1 (what to draw), we still need to have question 3 answered. The code below becomes thus a necessity (actually, you are not forced to specify the positioning of an object, because its draw method positions it to (0,0) by default):

```
% question 1: what to draw
Picture.pict("foo");

% question 3: where to draw
pict.nw = (10,10);

% now we can draw
drawObject(pict);
```

How about question 2, how to draw? By default, this problem is addressed behind the scenes by the component. This means, for the Picture object, that a native picture is created from the given string, and around that picture certain margins are placed, by means of MetaPost equations. (The margins come in handy when one wants to quickly place Picture objects near others, so that the result doesn't look too cluttered.) If these equations were defined within the Picture constructor, then an usability problem would have appeared, because it wouldn't have been possible to modify the margins, as in the code below:

```
% question 1: what to draw
Picture.pict("foo");

% question 2: how to draw
pict.info.left := 10;
pict.info.boxed := 1;

% question 3: where to draw
pict.nw = (0,0);
```

```
% now we can draw
drawObject(pict);
```

To allow for this type of code, the equations that define the layout of the `Picture` object (here, what the margins are) must be defined somewhere after the constructor. This is done by a macro called `Picture_layout`. This macro defines all the equations which link the "what to draw" information to the "how to draw" information (which in our case is taken from the `info` member, a copy of `iPict`). Nevertheless, notice that `Picture_layouts` is not explicitly invoked. To the user's great relief, this is taken care of automatically within the `Picture_draw` macro.

There are times however, when explicitly invoking a macro like `Picture_layout` becomes a necessity. This is because, by contract, it is only after the `layout` macro is invoked that the final dimensions (width, height) of an object are definitely and permanently known. Imagine that we have a component whose job is to surround in a red-filled rectangle some other objects. This component needs to know what the dimensions of the contained objects are, in order to be able to set its own dimensions. At drawing time, the contained objects must not have been drawn already, because the red rectangle of the container would overwrite them. Therefore, the whole pseudo-code would be:

```
Create objects o1, o2, ... ok;
Create container c(o1, o2, ..., ok);
Optional: modify info-s for o1, o2, ... ok;
Optional: modify info for c;

layout c, requiring layout of o1, o2, ... ok;
establish where to draw c;
draw red rectangle defined by c;
draw components o1, o2, ...ok within c
```

Note that an object mustn't be laid out more than once, because otherwise inconsistent or superfluous equations would arise. To enforce this, by contract, any object must keep record of whether its layout method has already been invoked, and if the answer is affirmative, subsequent invocations of the layout macro would do nothing. It is very important to mention that after the `layout` macro is invoked over an object, modifying the `info` member of that object has no subsequent effect, since the layout equations are declared and interpreted only once.

Notes on the Implementation of Links

MetaUML considers edges in diagram graphs as links. A link is composed of a path and the heads (possible none, one or two). For example, an association has no heads, and one must simply draw along the path with a solid pen. An unidirectional aggregation has a solid path and two heads: one is an arrow and the other is a diamond. So the template algorithm for drawing a link is:

AB	the path specified by the user
$\lvert AA'\rvert$	`iLink.widthA`
$\lvert BB'\rvert$	`iLink.widthB`

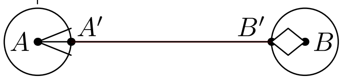


Figure 30. Details on how a link is drawn by MetaUML.

```
0. Reserve space for heads
1. Draw the path (except for the heads)
2. Draw head 1
3. Draw head 2
```

Each of the UML link types define how the drawing should be done, in each of the cases (1, 2 and 3). Consider the link type of unidirectional composition. Its "class" is declared as:

```
vardef CompositionUniInfo@# =
  LinkInfo@#;

  @#widthA      = defaultRelationHeadWidth;
  @#heightA     = defaultRelationHeadHeight;
  @#drawMethodA = "drawArrow";

  @#widthB      = defaultRelationHeadWidth;
  @#heightB     = defaultRelationHeadHeight;
  @#drawMethodB = "drawDiamondBlack";

  @#drawMethod = "drawLine";
enddef;
```

Using this definition, the actual description is created like this:

```
CompositionUniInfo.compositionUni;
```

As shown previously, is is the macro `link` which performs the actual drawing, using the link description information which is given as parameter (generally called `iLink`). For example, we can use:

```
link(aggregationUni)((0,0)--(40,0));
```

Let us see now the inner workings of macro `link`. Its definition is:

```
vardef link(text iLink)(expr myPath)=
   LinkStructure.ls(myPath,
                    iLink.widthA, iLink.widthB);
   drawLinkStructure(ls)(iLink);
enddef;
```

First, space is reserved for heads, by "shortening" the given path `myPath` by `iLink.widthA` at the beginning and by `iLink.widthB` at the end. After that, the shortened path is drawn with the "method" given by `iLink.drawMethod` and the heads with the "methods" `iLink.drawMethodA` and `iLink.drawMethodB`, respectively (figure 30).

Object Definitions: Easier generic_declare

In MetaPost if somebody wants to define something resembling a class, say Person, he would do something like this:

```
vardef Person@#(expr _name, _age)=
  % @# prefix can be seen as ‘this‘ pointer
  string @#name;
  numeric @#age;

  @#name := _name;
  @#age := _age;
enddef;
```

This allows for the creation of instances (or objects) of class Person by using declarations like:

```
Person.personA;
Person.personB;
```

However, if one also wants to able able to create indexed arrays of persons, such as Person.student0, Person.student1 etc., the definition of class Person must read:

```
vardef Person@#(expr _name, _age)=
  _n_ := str @#;
  generic_declare(string) _n.name;
  generic_declare(numeric) _n.age;

  @#name := _name;
  @#age := _age;
enddef;
```

This construction is rather inelegant. MetaUML offers alternative macros to achieve the same effect, uncluttering the code by removing the need for the unaesthetic _n_ and _n.

```
vardef Person@#(expr _name, _age)=
  attributes(@#);
  var(string) name;
  var(numeric) age;

  @#name := _name;
  @#age := _age;
enddef;
```

Customization in MetaUML: Examples

We have seen that in MetaUML the “how to draw” information is memorized into the so-called “Info” structures. For example, the default way in which a Picture object is to be drawn is recorded into an instance of PictureInfo, named iPict. In this section we present a case study involving the customization of Class objects. The customization of any other MetaUML objects works similarly. Here we cannot possibly present all the customization options for all kinds of MetaUML objects: this would take too long. Nevertheless, an interested reader can refer to the top of the appropriate MetaUML library file, where Info structures are defined. For example, class diagram related definitions are in metauml_class.mp, activity diagram definitions are in metauml_activity.mp etc.

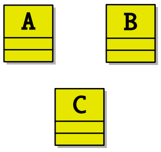


Figure 31. Changing the default settings for all classes.

Global settings

Let us assume that we do not particularly like the default foreground color of all classes, and wish to change it so something yellowish. In this scenario, one would most likely want to change the appropriate field in iClass:

```
iClass.foreColor := (.9, .9, 0);
```

After this, the following code produces the result in figure 31:

```
Class.A("A")()();
Class.B("B")()();
Class.C("C")()();

B.w = A.e + (20,0);
C.n = .5[A.se, B.sw] + (0, -10);

drawObjects(A, B, C);
```

Individual settings

When one wants to make modifications to the settings of one particular Class objects, another strategy is more appropriate. How about having class C stand out with a light blue foreground color, a bigger font size for the class name and a blue border (figure 32)?

```
iPict.foreColor := (.9, .9, 0);

Class.A("A")()();
Class.B("B")()();
Class.C("C")()();
C.info.foreColor := (.9, .7, .7);
C.info.borderColor := green;
C.info.iName.iFont.scale := 2;

% positioning code ommited
drawObjects(A, B, C);
```

As an aside, note that for each Class object its info member is created as a copy of iClass: the actual

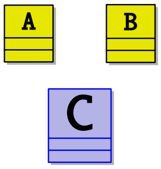


Figure 32. Individual customization of a class object.

drawing is performed using this copied information. Because of that, one can modify the info member after the object has been created and still get the desired results.

Another thing worth mentioning is that the ClassInfo structure contains the iName member, which is an instance of PictureInfo. In our example we do not want to modify the spacings around the Picture object, but the characteristics of the font its contents is typeset into. To do that, we modify the iName.iFont member, which by default is a copy of iFont (an instance of FontInfo, defined in util_picture.mp). If, for example, we want to change the font the class name is rendered into, we would set the attribute iName.iFont.name to a string representing a font name on our system (as used with the MetaPost infont operator).

Predefined settings

The third usage scenario is perhaps more interesting. Suppose that we have two types of classes which we want to draw differently. Making the setting adjustments for each individual class object would soon become a nuisance. MetaUML's solution consists in the ability of using predefined "how to draw" Info objects. Let us create such objects:

```
ClassInfoCopy.iHome(iClass);
iHome.foreColor := (0, .9, .9);

ClassInfo.iRemote;
iRemote.foreColor := (.9, .9, 0);
iRemote.borderColor := green;
```

Object iHome is a copy of iClass (as it might have been set at the time of the macro call). Object iRemote is created just as iClass is originally created. We can now use these Info objects to easily set the "how to draw" information for classes. The result is depicted in figure 33, please note the "E" prefix in EClass:

```
EClass.A(iHome)("UserHome")()();
EClass.B(iRemote)("UserRemote")()();
EClass.C(iHome)("CartHome")()();
EClass.D(iRemote)("CartRemote")()();
```

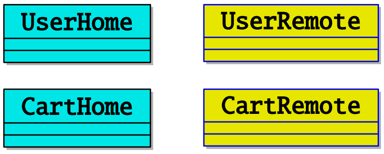


Figure 33. Using predefined settings.

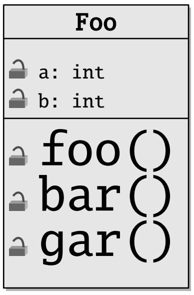


Figure 34. Extreme customization of a class. You may want not to do this, after all.

Extreme customization

When another font (or font size) is used, one may also want to modify the spacings between the attributes' and methods' baselines. Figure 34 is the result of the (unlikely) code:

```
Class.A("Foo")
  ("a: int", "b: int")
  ("foo()", "bar()", "gar()");
A.info.iAttributeStack.iPict.iFont.scale := 0.8;
A.info.iAttributeStack.top := 10;
A.info.iAttributeStack.spacing := 11;

A.info.iMethodStack.iPict.iFont.scale := 2;
A.info.iMethodStack.spacing := 17;
A.info.iMethodStack.bottom := 10;

drawObject(A);
```

Both iAttributeStack and iMethodStack are instances of PictureStackInfo, which is used to control the display of PictureStack objects.

Conclusions

MetaUML is a GNU GPL library for typesetting UML diagrams, particularly useful in a TeX or LaTeX environment; see Knuth (1986), Lamport (1994). It provides an easy to use, human readable API.

The code of a diagram typeset in MetaUML appears clearer (at least to the author of this paper) than the corresponding code in uml.sty, pst-uml.sty, umldoc or even XMI; see Gjelstad (2001), Diamantini (1998), Palmer (1999), OMG (2003). It is the next best thing to using a visual tool, while having the freedom of not becoming technologically dependent of any particular visual tool.

The `util` infrastructure of MetaUML offers means of defining and using “objects”, which may recommend it for other typesetting projects, unrelated to UML. We mention here a few of its benefits: the ability to stack and align text in a visually pleasing way; a fine degree of control of how elements are laid out; the ability to group objects while having access to the properties of inner elements; a design pattern and syntactic sugar for writing modern-looking, reusable MetaPost code.

With this infrastructure in place, it should be possible to extend MetaUML until it offers complete UML 2.0 support.

References

Roegel, D. (2002). The METAOBJ tutorial and reference manual. Available from `www.loria.fr/ roegel/TeX/momanual.pdf`.

Knuth, D. E. (1986). *The TeXbook*. Addison-Wesley Publishing Company.

Lamport, L. (1994). *LaTeX a Document Preparation System*. Addison-Wesley Publishing Company, 2nd edition.

Gheorghies, O. (2005). MetaUML: Tutorial, Reference and Test Suite. Available from `http://metauml.sourceforge.net`.

Hobby, J. (1992) A User’s Manual for MetaPost. Available from `http://www.tug.org/tutorials/mp/`.

Gjelstad, E. (2001). uml.sty 0.09.09. Available from `http://heim.ifi.uio.no/~ellefg/uml.sty/`.

Diamantini, M. (1998). Interface utilisateur du package pst-uml. Available from `http://perce.de/LaTeX/pst-uml/`.

Palmer, D. (1999). The umldoc UML Documentation Package. Available from `http://www.charvolant.org/~elements/`.

Object Management Group (2003). XML Metadata Interchange (XMI) Specification. Available from `http://www.omg.org/`.

Ovidiu Gheorghieș
Faculty of Computer Science
“Al. I. Cuza” University of Iași
Romania
ogh@info.uaic.ro