MAPS

NUMMER 34 • NAJAAR 2006

NEDERLANDSTALIGE TEX GEBRUIKERSGROEP

Voorzitter
Hans Hagen
ntg-president@ntg.nl

Secretaris
Willi Egger
ntg-secretary@ntg.nl

Penningmeester
Wybo Dekker
ntg-treasurer@ntg.nl

Bestuursleden
Karel Wesseling
k.h.wesseling@planet.nl

Taco Hoekwater
taco@elvenkind.com

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 1.40-beta 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 34e nummer van de Maps; veel nieuwe ontwikkelingen deze keer, en veel verrassingen:

□ Taco Hoekwater *(Nog een laatste dingetje …)*: Hebt u een Mac? Kom dan in actie! Want zonder u wordt uw TEX daarop niet meer ververst, omdat Gerben Wierda nu wel eens iets anders wil gaan doen.

□ Hans Hagen *(What tools do ConTEXt users have?)*: ConTEXt is zo langzamerhand omringd door vele Perl-, Ruby- en andere tools. Voor het automatisch laten lopen van ConTEXt-runs, voor het zoeken van files, voor het installeren van fonts, en nog veel meer. Ook voor niet-ConTEXt-gebruikers zitten daar interessante tools bij, dus haak niet meteen af als u het woord ConTEXt tegenkomt.

□ Taco en Hans *(International User Meeting 2006)*: De eerste internationale ConTEXt-meeting komt eraan. De plaats van handeling is Epen. Ik wist zelf niet waar dat ligt, het kaartje is ook niet erg gedetailleerd, maar het blijkt een een steenworp afstand van het drielandenpunt te liggen. De koeienletters waarin de titel van dit verhaal is gezet komen verderop in deze Maps nog aan bod.

□ Hans Hagen *(MKII – MKIV)*: De cryptische titel blijkt te verwijzen naar ConTEXt-versienummers, en vertelt ons dat ConTEXt van Mark II naar Mark IV gaat. Dat suggereert dat Hans niet kan tellen, maar lees het eerst maar en steek veel op over de nieuwste ontwikkelingen in TEX in het algemeen en in ConTEXt in het bijzonder.

□ Aditya Mahajan *(Display Math in ConTEXt)*: LaTEX-ers onder de wiskundigen zijn erg gelukkig met het amstex package, maar ze kunnen niet langer een lange neus trekken naar ConTEXt-ers, want hier wordt beschreven hoe dezelfde faciliteiten onder ConTEXt beschikbaar zijn gekomen.

□ Taco Hoekwater *(Metapost Developments)*: Er is een nieuwe release van MetaPost uitgekomen. Taco vertelt over de stormachtige ontwikkelingen.

□ Jerzy Ludwichowski *(Invitation to EuroTEX2007)*: Een vreemde eend in de bijt, voor een keer: geen artikel, maar een uitnodiging om naar EuroTEX2007 in Bachotek, Polen te komen. Alle hiervoor genoemde en hierna nog te noemen nieuwe ontwikkeling komen daar aan de orde.

□ Bogusław Jackowski *(Appendix G illuminated)* geeft, voor de Plain-ers onder ons, een heldere uitleg over Appendix G: *Generating Boxes from Formulas* van het TEXbook.

□ Hans Hagen, Jerzy Ludwichowski, Volker Schaa *(The New Font Project: TEX Gyre)* Nieuwe ontwikkelingen: steeds meer vrije fonts komen ons ter beschikking.

□ Taco Hoekwater, Hans Hagen *(The making of a (TEX) font)* beschrijven hoe de al eerder genoemde koeienletters tot stand kwamen; erg leerzaam voor wie zelf eens met fonts aan de gang wil.

□ Paul Lemmens *(Proefschrift in LaTeX)* zet uiteen hoe zijn proefschrift een queeste was, niet alleen op wetenschappelijk, maar ook op TEX-gebied.

□ Hans van der Meer *(Random bit generator in TEX)*: Weer eens heel iets anders: scripting in TEX!

□ Paweł Jackowski *(Enjoy TEX pearls diving!)*: Nee, met Perl heeft het niet te maken, maar pareltjes zijn het wel: veel handige en leerzame TEX-macro's.

□ Siep Kroonenberg *(Epspdf)* vertelt over haar Rubytrukendoos waarmee velerlei conversies tussen PDF en PostScript kunnen worden gerealiseerd, zowel heen als terug.

□ Frans Goddijn *(David Walden interview)* ondervraagt een bekende ondervrager in TEX-land.

□ Wybo Dekker *(The isodoc class)* beschrijft een class die voortbouwt op de NTG brief *class*. Via een *key=value* mechanisme is die gemakkelijker in het gebruik en uitbreidbaar voor andere documenttypen dan de nu al beschikbare brieven en facturen.

□ Geert C.H.M. Verhaag *(Creating a Dust-cover in ConTEXt)* beschrijft hoe hij in LaTEX een sierlijk boekomslag ontwierp.

□ Taco Hoekwater *(TUG 2006 Report)* geeft verslag van de TUG-conferentie in Marrakesh.

De redactie wenst u weer veel inspiratie en veel leesplezier.

PS. Dante 2007 is gepland voor 7-9 maart 2007 in Münster/Westfalen. De Dante voorjaarsbijeenkomsten hebben vaak een wat internationaal karakter. Omdat de bijeenkomst dit keer niet ver van onze grens plaatsvindt is hij wellicht ook voor NTG leden interessant.

Wybo Dekker, Taco Hoekwater

Nog een laatste dingetje . . .

. . . *de i-Installer zoekt een nieuwe beheerder*

I Quit

Gerben Wierda en Renée Van Roode gaven op de TUG bijeenkomst een gezamenlijke presentatie met de titel 'Life with TEX'. Over TUG vindt u elders in deze MAPS meer, maar het belangrijkste onderdeel van deze presentatie was de laatste slide, met als titel: 'I Quit'.

Voor Gerben is het mooi geweest. Na zes jaar van programmeren aan de i-Installer, het aanmaken en bijwerken van i-Packages, en vooral het ondersteunen van vaak volstrekt on-technische gebruikers, is het nu tijd voor hem om TEX echt zelf te gaan gebruiken.

Natuurlijk komt zo'n beslissing niet zomaar: hij gaf aan dat met name de toegenomen werkdruk de doorslag geeft. Die extra werkdruk wordt bijvoorbeeld veroorzaakt door de recente overgang van het relatief kleine en stabiele teTEX naar de veel grotere en dynamischere TEX–live distributie.

Een noodzaak, omdat teTEX niet langer wordt onderhouden of doorontwikkeld. Maar het was een heleboel werk, omdat de beide distributies flink van elkaar verschillen. En doordat TEX–live groter is en sneller verandert, zal er meer werk gedaan moeten worden dan vroeger om de i-Packages te onderhouden.

Aan het einde van dit jaar zal er waarschijnlijk nog één update van de i-Installer komen, met ondersteuning voor beveiligde proxy servers en uitgebreidere documentatie. Daarna zal Gerben hooguit nog die i-Packages updaten die hij zelf gebruikt, maar zelfs dat is niet zeker. Met ingang van 1 januari 2007 is het zeer zeker afgelopen met de jarenlange e-mail ondersteuning door Gerben.

Er is niet alleen een programma dat zoekt naar een nieuwe beheerder–ontwikkelaar (i-Installer), maar ook een complete TEX distributie (gwTEX) met hetzelfde probleem. Tenzij er iemand (of misschien een groepje mensen) bereid is het werk van Gerben over te nemen, is het straks afgelopen met de vrijwel perfecte ondersteuning van TEX onder Mac OS X.

Deze keer zal de TEX Collection nog een gwTEX bevatten . . . Of de volgende dat ook nog zal hebben, hangt mede van u af.

Taco Hoekwater

What tools do ConTEXt users have?

Abstract
This article describes the tools that are available for running ConTEXt, for testing, finding, analysing files, installing fonts, and more.

A bit of history

When we started working on ConTEXt, MS Windows (and before that 4Dos) was our main platform; it still is for development (we use Unix on the web and file servers and the Mac for fun). So, when ConTEXt was integrated into the TEX distributions we faced the problem of portability. Since one needs auxiliary programs[1] *e.g.* for sorting an index, we had written TEXutil, and the lack of a command line handler made us come up with TEXexec. Both were written in Modula but were rewritten in Perl in order to be usable on platforms other than MS Windows. It was easier to maintain a Perl version than to deal with low-level platform issues indefinitely.

As both our own and user demands grew, we wrote more tools and found out that they could best be written in Ruby. In the meantime TEXexec has been rewritten in Ruby, and relevant parts of TEXutil have been merged into it.

Launching scripts

Starting a script on an MS Windows box can be done using a so-called stub, a small program or command file with the same name that locates a similarly named script. On Unix some shell magic can be used to do the same or one can fall back on a magic preamble (a Bash/Perl mixture) fooling the operating system into locating and spawning the script using the right interpreter. By now, MS Windows has a convenient file association mechanism (but one has to activate it first) while Unix needs a (nowadays less path sensitive) shebang line and a suffix-less copy of the script.

Nevertheless we decided to come up with a less sensitive approach which also gave us the opportunity to accomplish a few more things: TEXMFstart. This script locates and executes a script (or program) in the TEX tree and executes it.

```
texmfstart texexec somefile.tex
```

When you incorporate TEX in work flows, calling TEXexec this way is rather future safe. Actually, because of this method, we could make the transition from TEXexec being a Perl script to being a Ruby program without too much trouble. A side effect of this way of launching scripts is that nested calls are faster because some information is passed on to child runs.

The script is also able to sort out a couple of things, for instance where files reside. Nowadays one will seldom use TEX alone and not all text processing (or related) programs have a clear concept of resource management and/or can work well with a TDS conforming tree.

```
texmfstart bin:xsltproc --output=new.xml \
  kpse:how.xsl old.xml
```

This[2] will locate the file `how.xsl` in the TEX tree and expand the file name to the full path. That way one can keep XSLT scripts organized as well. There are a few more such prefixes.

Other features are locating and showing documentation and launching editors with files located in the tree. The following call will open the `texmf.cnf` file that is currently used.

```
texmfstart --edit kpse:texmf.cnf
```

The script can initialize a tree so one can effectively run multiple trees in parallel. It does so by loading (when present) a file with variable specifications (more about that later).

```
texmfstart --tree=e:/tex-2003 \
  texexec somefile.tex
```

We often use a different tree for each project because commercial fonts may be project related and this way we can move a tree around without running into copyright problems (read: installing all fonts on each box).

```
texmfstart --tree=e:/tex-2003 \
  texexec somefile.tex
```

Another handy feature is conditional processing. In the following case the test file will only be processed when it has changed.

```
texmfstart --ifchanged=test.r --direct R \
  "-q --save --restore < test.r"
```

In a similar fashion one can make running dependent on time stamp comparison. More details can be found in the manual.

Managing ConTEXt runs

The TEXexec script manages a user's TEX run. There are many factors that influence such a run:

- Since ConTEXt uses the same format for all back ends, it depends on loading the relevant back end driver modules. Although one has complete control, life can be made easier when this is done automatically.
- A first pass may generate data needed in a successive pass. There may be references, tables of contents, indexes, etc. so we need a way to manage multiple runs. We have to make sure that neither fewer nor more runs than needed take place.
- A run may demand further action between runs, like graphic manipulations or delayed MetaPost execution.
- We may want to run different TEX engines, apply different back ends, use different user interfaces. Also, the name and way of calling TEX may change over time, something that we don't want users to be bothered with.
- We may want to process a TEX or XML file under different style regimes or enable style-specific modes.
- The document may need an additional page imposition pass, managed in such a way that no auxiliary data gets messed up.
- We may want to close and open the result in a viewer after the run is done.

This and a bit more is handled by TEXexec. When dealing with ConTEXt files the script will do a few things users are normally not aware of, like making sure that the random seed is frozen for a run, bugs in programs are caught (as long as needed) and that omissions in the `texmf.cnf` settings are compensated for. In addition TEXexec provides a few features for combining and manipulating PDF files.

The latest versions of TEXexec also support socalled ctx files. These are files in XML format that describe a process, additional pre- and post processing needed, styles and modules to be used, etc.[3] This means that one can easily configure projects without the need to tweak source files or editor setups or give explicit commands. Think of situations where an XML file (or bunch of files) has to be converted to another variant in order to be processed. TEXexec will only do that conversion when needed. In Figure 1 we show the file that is used in the MathAdore project.[4] The source file contains OpenMath and what we call 'shortcut math' and after normalizing this to OpenMath (first conversion) we convert the math to content MATHML (second conversion).

The source file contains a reference to this ctx file and when TEXexec is applied to the source file, it will take the appropriate actions. Such a reference looks like:

```
<?ctx-dir job ctxfile ../mathadore.ctx ?>
```

Here "`ctx-dir`" denotes a ConTEXt directive.

When dealing with a TEX file, TEXexec will scan the first line for comments that serve a similar purpose.

Handling the utility file

For a long time TEXutil was called from within TEXexec to handle the utility file that collects the index entries, tables of contents, references, etc. Nowadays this functionality is integrated in TEXexec which is more efficient. We also took the opportunity to enhance the sorting features so that one can mix language specific sorting rules.

The original TEXutil is also responsible for some other manipulations, like analyzing graphics. That kind of functionality has been moved to other scripts and more modern ways of dealing with such issues. Because we were in a transition stage to Ruby scripting, it was a good moment to say goodbye to TEXutil and concentrate on building a more extensive set of tools.

The tools collection

Instead of expanding TEXutil, we decided to spread functionality over multiple scripts. These can be recognized by their name: they all end with `tools`. If you call them using TEXMFstart there is not much opportunity for conflicts with existing tools.

Each tool comes with a manual, so we will not discuss details here.

ctxtools. This tool provides ConTEXt related features, like generating generic pattern files (so that we are independent), providing editor syntax checking files derived from the generic ConTEXt interface definition (handy for lexers), generating documentation (from the ConTEXt source code), updating ConTEXt (by downloading an archive and regenerating formats), etc.

rlxtools. The 'r' represents resources, normally graphics, the 'l' stands for libraries, and the 'x' (indeed) for XML. This tool can analyze graphic files and manipulate resources using other programs. For instance it can be used to down sample files at run time, to handle special color conversion, and to convert graphics to formats acceptable for TEX. By using the run time converters one can build work flows without the need to rely on additional scripting. There is a dedicated manual on this topic so we will not bore you here with yet another blob of XML.

```
<?xml version='1.0' standalone='yes'?>
<ctx:job>
    <ctx:message>mathadore</ctx:message>
    <ctx:preprocess suffix='prep'>
        <ctx:processors>
            <ctx:processor name='openmath' suffix='om'>texmfstart
                --direct xsltproc
                --output <ctx:value name='new'/>
                kpse:x-sm2om.xsl <ctx:value name='old'/>
            </ctx:processor>
            <ctx:processor name='mathadore' suffix='prep'>texmfstart
                --direct xsltproc
                --output <ctx:value name='new'/>
                kpse:x-openmath.xsl
                <ctx:value name='old'/>.om
            </ctx:processor>
        </ctx:processors>
        <ctx:files>
            <ctx:file processor='openmath,mathadore'>v*.xml</ctx:file>
            <ctx:file processor='openmath,mathadore'>h*.xml</ctx:file>
            <ctx:file processor='openmath,mathadore'>openmath*.xml</ctx:file>
        </ctx:files>
    </ctx:preprocess>
    <ctx:process>
        <ctx:resources>
            <ctx:environment>o-m4all.tex</ctx:environment>
        </ctx:resources>
    </ctx:process>
    <ctx:postprocess>
    </ctx:postprocess>
</ctx:job>
```

Figure 1. A ctx file used in the MathAdore project

xmltools. You can use this tool to do a simple analysis on an XML file. Another option is to generate a directory listing in XML format. In both cases, the result can be fed into ConTEXt and used in the process. A more obscure option is to generate images from MATHML snippets. This script will without doubt include more features in the future.

pdftools. This is work in progress. One can for instance roughly analyze PDF files. It also provides a way to manipulate colors in PDF images but that feature is now supported in ConTEXt directly.

textools. Users will seldom need this tool. It can fix things in a TDS compliant tree (for instance when the standard has changed), it deals with a few cross platform issues, it can help you to create so-called TPM archives (and is meant for ConTEXt module writers) and it can merge updates into your tree.

mpstools. In the future this tool will host the now standalone MetaPost to PDF wrapper (mptopdf) as well as the cropper (both are still Perl scripts).

tmftools. This script encapsulates some of the functionality of the Ruby based kpsewhich functionality that we use. In the future we may completely move away from the binary because the script is just as fast or faster when it serializes the database. The script can act as a kpsewhich server. The script can also analyze the tree for duplicates.

runtools. Because TEX is multi platform and because we (need to) run services on multiple platforms, we use this script to do things normally done at the console (shell). It just loads the given Ruby scripts with the appropriate library. We also use this tool to generate the ConTEXt distribution.

exatools. This is a more obscure tool. It provides some features related to form based style control and web driven TEX processing that we use in projects.

pstopdf. This last one is not a collection like the previous tools. It started long ago as a wrapper for Ghostscript. It still provides this function and over the years we've added quite a bit of filtering to it (we just filter the things that Ghostscript fails on or gets confused from). In the meantime we cheat on the name since it also manages the conversion of bitmap images, especially cached down sampling, using ImageMagick as

```
# file   : setuptex.tmf (the less generic version have suffixes like cmd, sh, csh etc)
# author : Hans Hagen - PRAGMA ADE - Hasselt NL - www.pragma-ade.com
# usage  : texmfstart --tree=f:/minimal/tex ...
#
# this assumes that calling script sets TEXPATH without a trailing
# slash; %VARNAME% expands to the environment variable, $VARNAME
# is left untouched; we also assume that TEXOS is set.

TEXMFMAIN    = %TEXPATH%/texmf
TEXMFLOCAL   = %TEXPATH%/texmf-local
TEXMFFONTS   = %TEXPATH%/texmf-fonts
TEXMFPROJECT = %TEXPATH%/texmf-project
VARTEXMF     = %TMP%/texmf-var
HOMETEXMF    =

TEXMFOS      = %TEXPATH%/%TEXOS%

TEXMFCNF     = %TEXPATH%/texmf{-local,}/web2c
TEXMF        = {$TEXMFOS,$TEXMFPROJECT,$TEXMFFONTS,$TEXMFLOCAL,!!$TEXMFMAIN}
TEXMFDBS     = $TEXMF

TEXFORMATS   = %TEXMFOS%/web2c/{$engine,}
MPMEMS       = %TEXFORMATS%
TEXPOOL      = %TEXFORMATS%
MPPOOL       = %TEXPOOL%

PATH         > %TEXMFOS%/bin
PATH         > %TEXMFLOCAL%/scripts/perl/context
PATH         > %TEXMFLOCAL%/scripts/ruby/context

TEXINPUTS    =
MPINPUTS     =
MFINPUTS     =
```

Figure 2. Example `texmf.cnf` file

well as conversion from SVG to PDF using Inkscape.

texfont. This script has been around for a while now and is used to install (commercial) fonts. It generates metric files, map files, and a demo file so that one can see if things went right. ConTEXt does not depend on (ever changing) map file methods and loads map files on demand. You can generate map files for dvipdfmx with the previously mentioned `ctxtools`.

More

There are a few more scripts, like `concheck` (simple syntax checking) and `texsync` (synchronizing minimal distributions) but we will not discuss them here.

Integration

When setting up multiple TEX trees, the trick is in isolating them as much as possible. Because one can never be sure how distributions set things up, we revert to setting environment variables, which will then take precedence over the settings in a regular `texmf.cnf` file. In the TEXMFstart manual you can find more details on how we take care of this; here we only show an example of such an file in Figure 2.

When the tree flag is given, TEXMFstart will read this file and set the environment variables accordingly before it launches the program it is supposed to start. In fact, a tree specification can specify a file, but by default the `setyptex` one is taken.

```
texmfstart \
  --tree=f:/minimal/tex/setuptex.tmf \
  texexec test.tex
```

Since TEXMFstart can load several such files, we can also use this method to preset more environment variables, for instance pointers to resources like graphics. This is what the `--env` or `--environment` option is for, as in:

```
texmfstart --tree=f:/minimal/tex \
  --env=xyz.tmf texexec test.tex
```

The advantage of this variable setting game is that instead of cooking up scripts with statements like:

```
thread.new do
 ENV["something"] = "nothing"
 a = "texmfstart --tree=f:/minimal/tex --"
 system(a+"env=xyz.tmf texexec test.tex")
end
```

we can put the variable definition in a file and say:

```
thread.new do
 a = "texmfstart --tree=f:/minimal/tex --"
 system(a+"env=xyz.tmf texexec test.tex")
end
```

This has not only a big advantage in terms of isolation (and maintenance) but is also more robust since one can never be sure if another thread is not setting the same variable too, thereby creating much confusion for all the other threads that use the same variable. Since TEXMFstart runs as a separate process, it can set its variables independently.

Whenever (on the ConTEXt mailing list) you see mentioning of something named `setuptex`, you can be sure that it relates to initializing a TEX tree (probably a minimal ConTEXt tree) in an isolated way.

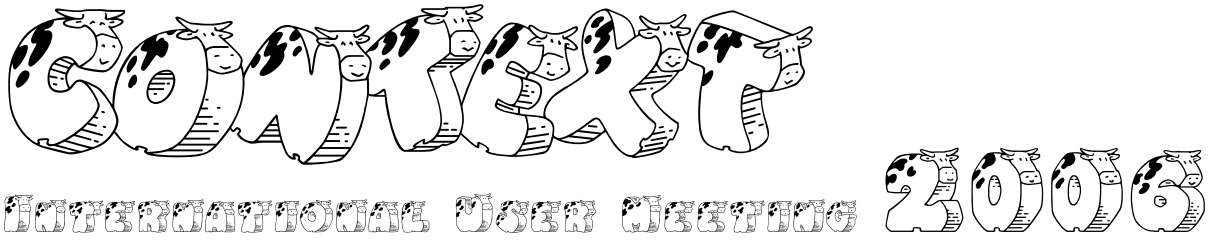
Conclusion

In this short article we have tried to give you an impression of what is needed in order to make TEX usable in a diversity of today's environments. It was not our intention to be complete, because for that purpose we have manuals. One thing should be made clear: although TEX itself is pretty stable, the same cannot be said for the environment that it is used in. Just telling TEX to process a file is not enough nowadays. This also means that ConTEXt and its tools, in order to keep up, need to be adapted to current needs. On the other hand, by organizing the functionality in tools, and by using a modern and reliable scripting language like Ruby, users don't pay a high price for this. Most nasty details can be hidden from them.

Notes

1. We will use the terms 'scripts' and 'programs' interchangeably.
2. The backslash at the end of line denotes a continued line.
3. Although one can use the `ctx` suffix for ConTEXt related TEX files, this is normally a bad idea.
4. This project will provide highly interactive math to schools and is conducted in cooperation with the University of Eindhoven.

Hans Hagen



The first international ConTEXt user meeting will take place in Epen, The Netherlands. The meeting will start on Friday March 23 at dinnertime, and will finish on Monday morning.

For those of you who are already familiar with ConTEXt, this meeting is an excellent opportunity to learn some new tricks, meet fellow users from the ConTEXt mailing list, talk to developers and hackers, and promote your feature request list.

If you are not already using ConTEXt, this is the ideal place to acquaint yourself with our beautiful macro package and meet your potential new friends !

Call for papers

The programme is not finalized yet, we welcome contributions on all levels of expertise. Please drop us a line if you are willing to give a presentation or tutorial.

Call for topics

Please *also* let us know if there is a specific topic you would like to hear more about.

Confirmed presentations

- Thomas A. Schmitz: Typesetting greek with ConTEXt
- Hans Hagen: ConTEXt IV and MetaTEX
- Idris Samawi Hamid: Al Sanaah: arabic typesetting
- Taco Hoekwater: Linguistic macros
- Jean-Michel Hufflen: MlBibTeX
- Sanjoy Mahajan: Typesetting a physics textbook with ConTEXt

Confirmed tutorials

- Willi Egger: Page layout and arrangements
- Taco Hoekwater: Creating a module

Workshops and discussions

ConTEXt documentation, Regression test suite, Development fund, Merchandising.

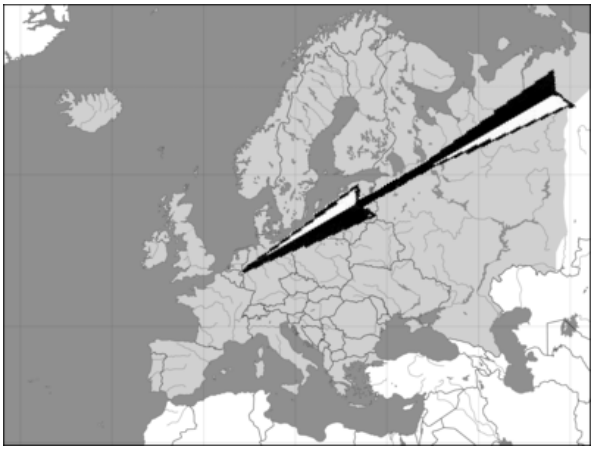
Registration

Because the organizers like the intimacy of small meetings – where not everything has to be pre-arranged and scheduled – we have picked a location that can hold only about three dozen participants. So, if you want to be sure there is place left, you should *register now*!

To register, please fill in the registration form on the website

http://context.aanhet.net/epen2007/

The base registration fee is 200 euro (spanning three nights and all meals from dinner on friday to breakfast on monday). Discounts are available for TEX user group members and for people that spend the nights away from the conference location.



Hope to see you there!

Taco and Hans

MKII – MKIV

Abstract
This article is the first in a series about ConTEXt and LuaTEX. For those who use ConTEXt it is a progress report of the development process and the choices that are being made. For those not using ConTEXt it gives some insight in what LuaTEX is about.

From Mark II to Mark IV

Sometime in 2005 the development of LUATEX started, a further development of PDFTEX and a precursor to PDFTEX version 2. This TEX variant will provide:

- 21–32 bit internals plus a code cleanup
- flexible support for OpenType fonts
- an internal UTF data flow
- the bidirectional typesetting of ALEPH
- LUA callbacks to the most relevant TEX internals
- some extensions to TEX (for instance math)
- an efficient way to communicate with MetaPost

In the tradition of TEX this successor will be downward compatible in most essential parts and in the end, there is still PDFTEX version 1 as fall back.

In the mean time we have seen another unicode variant show up, XETEX which is under active development, uses external libraries, provides access to the fonts on the operating system, etc.

From the beginning, ConTEXt always worked with all engines. This was achieved by conditional code blocks: depending on what engine was used, different code was put in the format and/or used at runtime. Users normally were unaware of this. Examples of engines are ε-TEX, ALEPH, and XETEX. Because nowadays all engines provide the ε-TEX features, in August 2006 we decided to consider those features to be present and drop providing the standard TEX compatible variants. This is a small effort because all code that is sensitive for optimization already has ε-TEX code branches for many years.

However, with the arrival of LUATEX, we need a more drastic approach. Quite some existing code can go away and will be replaced by different solutions. Where TEX code ends up in the format file, along with its state, LUA code will be initiated at run time, after a LUA instance is started. ConTEXt reserves its own instance of LUA.

Most of this will go unnoticed for the users because the user interface will not change. For developers however, we need to provide a mechanism to deal with these issues. This is why, for the first time in ConTEXt's history we will officially use a kind of version tag. When we changed the low level interface from Dutch to English we jokingly talked of version 2. So, it makes sense to follow this lead.

- ConTEXt Mark I At that moment we still had a low level Dutch interface, invisible for users but not for developers.
- ConTEXt Mark II We now have a low level English interface, which (as we indeed saw happen) triggers more development by users.
- ConTEXt Mark IV This is the next generation of ConTEXt, with parts re–implemented. It's an at some points drastic system overhaul.

Keep in mind that the functionality does not change, although in some places, for instance fonts, Mark IV may provide additional functionality. The reason why most users will not notice the difference (maybe apart from performance and convenience) is that at the user interface level nothing changes (most of it deals with typesetting, not with low level details).

The hole in the numbering permits us to provide a Mark III version as well. Once XeTeX is stable, we may use that slot for XeTeX specific implementations.

As per August 2006 the banner is adapted to this distinction:

```
...  ver: 2006.09.06 22:46 MK II  fmt: 2006.9.6  ...
...  ver: 2006.09.06 22:47 MK IV  fmt: 2006.9.6  ...
```

This numbering system is reflected at the file level in such a way that we can keep developing the way we do, i.e. no files all over the place, in subdirectories, etc.

Most of the system's core files are not affected, but some may be, like those dealing with fonts, input- and output encodings, file handling, etc. Those files may come with different suffixes:

- `somefile.tex`: the main file, implementing the interface and common code
- `somefile.mkii`: mostly existing code, suitable for good old TeX (ε-TeX, pdfTeX, Aleph).
- `somefile.mkiv`: code optimized for use with LuaTeX, which could follow completely different approaches
- `somefile.lua`: Lua code, loaded at format generation time and/or runtime

As said, some day `somefile.mkiii` code may show up. Which variant is loaded is determined automatically at format generation time as well as at run time.

How LUA fits in

introduction

Here I will discuss a few of the experiments that drove the development of LuaTeX. It describes the state of affairs around the time that we were preparing for TUG 2006. This development was pretty demanding for Taco and me but also much fun. We were in a kind of permanent Skype chat session, with binaries flowing in one direction and TeX and Lua code the other way. By gradually replacing (even critical) components of ConTeXt we had a real test bed and torture tests helped us to explore and debug at the same time. Because Taco uses Linux as platform and I mostly use MS Windows, we could investigate platform dependent issues conveniently. While reading this text, keep in mind that this is just the beginning of the game.

I will not provide sample code here. When possible, the Mark IV code transparantly replaces Mark II code and users will seldom notices that something happens in different way. Of course the potential is there and future extensions may be unique to Mark IV.

compatibility

The first experiments, already conducted with the experimental versions involved runtime conversion of one type of input into another. An example of this is the (TI) calculator math input handler that converts a rather natural math sequence into TeX and feeds that back into TeX. This mechanism eventually will evolve into a configurable math input handler. Such applications are unique to Mark IV code and will not be backported to Mark II. The question is where downward compatibility will become a problem. We don't expect many problems, apart from occasional bugs that result from splitting the code base, mostly because new features will not affect older functionality. Because we have to reorganize the code base a bit, we also use this opportunity to start making a variant of ConTeXt which consists of building blocks: MetaTeX. This is less interesting for the average user, but may be

of interest for those using ConTEXt in workflows where only part of the functionality is needed.

metapost

Of course, when I experiment with such new things, I cannot let MetaPost leave untouched. And so, in the early stage of LUATEX development I decided to play with two MetaPost related features: conversion and runtime processing.

Conversion from MetaPost output to PDF is currently done in pure TEX code. Apart from convenience, this has the advantage that we can let TEX take care of font inclusions. The tricky part of this conversion is that MetaPost output has some weird aspects, like DVIPS specific linewidth snapping. Another nasty element in the conversion is that we need to transform paths when pens are used. Anyhow, the converter has reached a rather stable state by now.

One of the ideas with MetaPost version 1^+ is that we will have an alternative output mode. In the perspective of LUATEX it makes sense to have a LUA output mode. Whatever converter we use, it needs to deal with METAFUN specials. These are responsible for special features like transparency, graphic inclusion, shading, and more. Currently we misuse colors to signal such features, but the new pre/post path hooks permit more advanced implementations. Experimenting with such new features is easier in LUA than in TEX.

The Mark IV converter is a multi–pass converter. First we clean up the MetaPost output, next we convert the PostScript code into LUA calls. We assume that this LUA code eventually can be output directly from MetaPost. We then evaluate this converted LUA blob, which results in TEX commands. Think of:

```
1.2 setlinejoin
```

turned into:

```
mp.setlinejoin(1.2)
```

becoming:

```
\PDFcode{1.2 j}
```

which is, when the PDFTEX driver is active, equivalent to:

```
\pdfliteral{1.2 j}
```

Of course, when paths are involved, more things happen behind the scenes, but in the end an `mp.path` enters the LUA machinery.

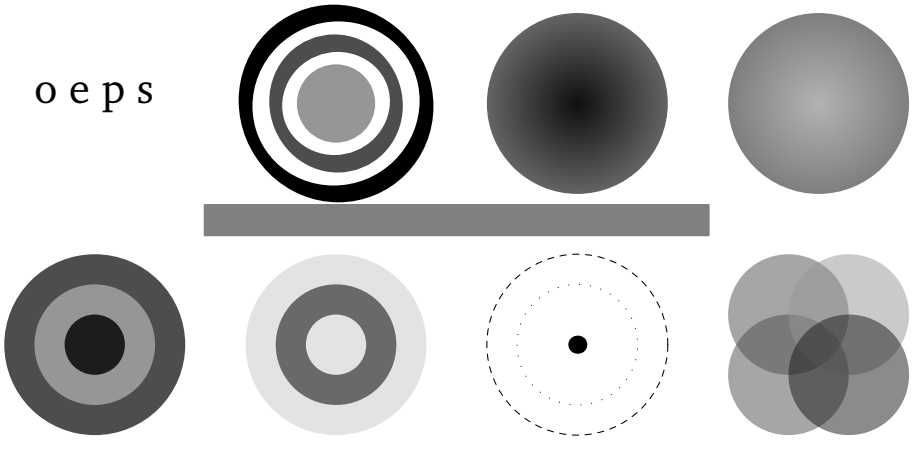


Figure 1. converter test figure

prologues/mpprocset	1/0	1/1	2/0 2/1
Mark II	8.5 (5.7)	8.0 (5.5)	8.8 8.5
Mark IV	16.1 (10.6)	7.2 (4.5)	16.3 7.4

Table 1. converter speed benchmark data

When the Mark IV converter reached a stable state, tests demonstrated then the code was upto 20% slower that the pure TEX alternative on average graphics, and but faster when many complex path transformations (due to penshapes) need to be done. This slowdown was due to the cleanup (using expressions) and intermediate conversion. Because Taco develops LUATEX as well as maintains and extends MetaPost, we conducted experiments that combine features of these programs. As a result of this, shortcuts found their way into the MetaPost output.

Cleaning up the MetaPost output using LUA expressions takes relatively much time. However, starting with version 0.970 MetaPost uses a preamble, which permits not only short commands, but also gets rid of the weird linewidth and filldraw related PostScript constructs. The moderately complex graphic that we use for testing (figure 1) takes over 16 seconds when converted 250 times. When we enable shortcuts we can avoid part of the cleanup and runtime goes down to under 7.5 seconds. This is significantly faster than the Mark II code. We did experiments with simulated LUA output from MetaPost and then the Mark IV converter really flies. The benchmark data are shown in table 1. The values on Taco's system are given between parenthesis.

The main reason for the huge difference in the Mark IV times is that we do a rigourous cleanup of the older MetaPost output in order avoid messy the messy (but fast) code that we use in the Mark II converter. Think of:

```
0 0.5 dtransform truncate idtransform setlinewidth pop
closepath gsave fill grestore stroke
```

In the Mark II converter, we push every number or keyword on a stack and use keywords as trigger points. In the Mark IV code we convert the stack based PostScript calls to LUA function calls. Lines as shown are converted to single calls first. When `prologues` is set to 2, such line no longer show up and are replaced by simple calls accompanied by definitions in the preamble. Not only that, instead of verbose keywords, one or two character shortcuts are used. This means that the Mark II code can be faster when procsets are used because shorter strings end up in the stack and comparison happens faster. On the other hand, when no procsets are used, the runtime is longer because of the larger preamble.

Because the converter is used outside CONTEXT as well, we support all combinations in order not to get error messages, but the converter is supposed to work with the following settings:

```
prologues := 1 ;
mpprocset := 1 ;
```

We don't need to set `prologues` to 2 (font encodings in file) or 3 (also font resources in file). So, in the end, the comparison in speed comes down to 8.0 seconds for Mark II code and 7.2 seconds for the Mark IV code when using the latest greatest MetaPost. When we simulate LUA output from MetaPost, we end up with 4.2 seconds runtime and when MetaPost could produce the converter's TEX commands, we need only 0.3 seconds for embedding the 250 instances. This includes TEX taking care of handling the specials, some of which demand building moderately complex PDF data structures.

But, conversion is not the only factor in convenient MetaPost usage. First of all, runtime MetaPost processing takes time. The actual time spent on handling

embedded MetaPost graphics is also dependent on the speed of starting up MetaPost, which in turn depends on the size of the TEX trees used: the bigger these are, the more time KPSE spends on loading the `ls-R` databases. Eventually this bottleneck may go away when we have MetaPost as a library. (In ConTEXt one can also run MetaPost between runs. Which method is faster, depends on the amount and complexity of the graphics.)

Another factor in dealing with MetaPost, is the usage of text in a graphic (`btex`, `textext`, etc.). Taco Hoekwater, Fabrice Popineau and I did some experiments with a persistent MetaPost session in the background in order to simulate a library. The results look very promising: the overhead of embedded MetaPost graphics goes to nearly zero, especially when we also let the parent TEX job handle the typesetting of texts. A side effect of these experiments was a new mechanism in ConTEXt (and METAFUN) where TEX did all typesetting of labels, and MetaPost only worked with an abstract representation of the result. This way we can completely avoid nested TEX runs (the ones triggered by MetaPost). This also works ok in Mark II mode.

Using a persistent MetaPost run and piping data into it is not the final solution if only because the terminal log becomes messed up too much, and also because intercepting errors is real messy. In the end we need a proper library approach, but the experiments demonstrated that we needed to go this way: handling hundreds of complex graphics that hold typeset paragraphs (being slanted and rotated and more by MetaPost), tooks mere seconds compared to minutes when using independent MetaPost runs for each job.

characters

Because LUATEX is UTF based, we need a different way to deal with input encoding. For this purpose there are callbacks that intercept the input and convert it as needed. For context this means that the regime related modules get a LUA based counterparts. As a prelude to advanced character manipulations, we already load extensive unicode and conversion tables, with the benefit of being able to handle case handling with LUA.

The character tables are derived from unicode tables and Mark II ConTEXt data files and generated using MTXTOOLS. The main character table is pretty large, and this made us experiment a bit with efficiency. It was in this stage that we realized that it made sense to use precompiled LUA code (using `luac`). During format generation we let ConTEXt keep track of used LUA files and compiled them on the fly. For a production run, the compiled files were loaded instead.

Because at that stage LUATEX was already a merge between PDFTEX and ALEPH, we had to deal with pretty large format files. About that moment the ConTEXt format with the english user interface amounted to:

date	**luatex**	**pdftex**	**xetex**	**aleph**
2006-09-18	9 552 042	7 068 643	8 374 996	7 942 044

One reason for the large size of the format file is that the memory footprint of a 32 bit TEX is larger than that of good old TEX, even with some of the clever memory allocation techniques as used in LUATEX. After some experiments where size and speed were measured Taco decided to compress the format using a level 3 ZIP compression. This brilliant move lead to the following size:

date	**luatex**	**pdftex**	**xetex**	**aleph**
2006-10-23	3 135 568	7 095 775	8 405 764	7 973 940

The first zipped versions were smaller (around 2.3 meg), but in the meantime we moved the LUA code into the format and the character related tables take some space.

debugging

In the process of experimenting with callbacks I played a bit with handling TEX error information. An option is to generate an HTML page instead of spitting out the usual blob of into on the terminal. In figure 2 and figure 3 you can see an example of this.

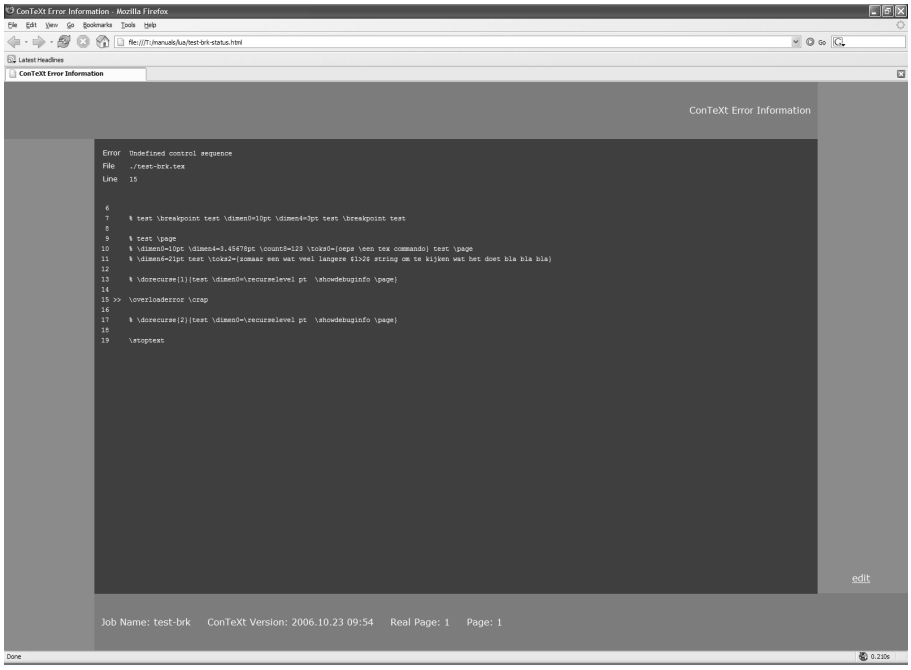


Figure 2. An example error screen.

Playing with such features gives us an impression of what kind of access we need to TEX's internals. It also formed a starting point for conversion routines and a mechanism for embedding LUA code in HTML pages generated by ConTEXt.

file io

Replacing TEX's in- and output handling is non–trival. Not only is the code quite interwoven in the WEB2C source, but there is also the KPSE library to deal with. This means that quite some callbacks are needed to handle the different types of files. Also, there is output to the log and terminal to take care of.

Getting this done took us quite some time and testing and debugging was good for some headaches. The mechanisms changed a few times, and TEX and LUA code was thrown away as soon as better solutions came around. Because we were testing on real documents, using a fully loaded ConTEXt we could converge to a stable version after a while.

Getting this IO stuff done is tightly related to generating the format and starting up LUATEX. If you want to overload the file searching and IO handling, you need overload as soon as possible. Because LUATEX is also supposed to work with the existing KPSE library, we still have that as fallback, but in principle one could think of a KPSE free version, in which case the default file searching is limited to the local path and memory initialization also reverts to the hard coded defaults. A complication is that the soure code has KPSE calls and references to KPSE variables all over the place, so occasionally we run into interesting bugs.

Anyhow, while Taco hacked his way around the code, I converted my existing RUBY based KPSE variant into LUA and started working from that point. The advantage of having our own IO handler is that we can go beyond KPSE. For instance,

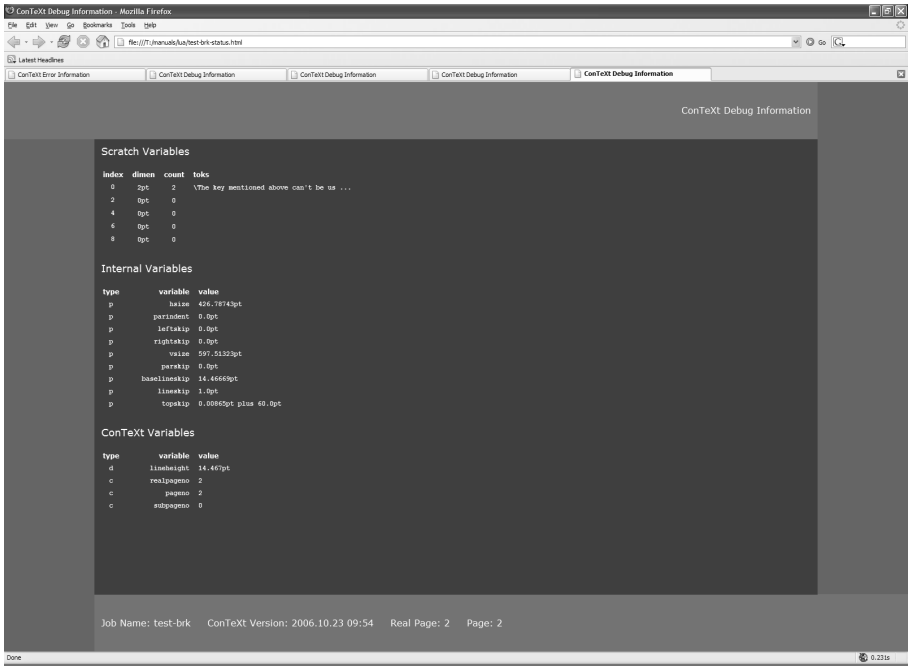


Figure 3. An example debug screen.

since LuaTEX has, among a few others, the ZIP libraries linked in, we can read from ZIP files, and keep all TEX related files in TDS compliant ZIP files as well. This means that one can say:

```
\input zip::somezipfile::somefile.tex
\input zip://somezipfile.zip/somepath/somefile.tex
```

and use similar references to access files. Of course we had to make sure that KPSE like searching in the TDS (standardized TEX trees) works smoothly. There are plans to link the curl library into LuaTEX, so that we can go beyong this and access repositories.

Of course, in order to be more or less KPSE and WEB2C compliant, we also need to support this paranoid file handling, so we provide mechanisms for that as well. In addition, we provide ways to create sandboxes for system calls.

Getting to intercept all log output (well, most log output) was a problem in itself. For this I used a (preliminary) XML based log format, which will make log parsing easier. Because we have full control over file searching, opening and closing, we can also provide more information about what files are loaded. For instance we can now easily trace what TFM files TEX reads.

Implementing additional methods for locating and opening files is not that complex because the library that ships with ConTEXt is already prepared for this. For instance, implementing support for:

```
\input http://www.someplace.org/somepath/somefile.tex
```

involved a few lines of code, most of which deals with caching the files. Because we overload the whole IO handling, this means that the following works ok:

```
\placefigure
  {http handling}
  {\externalfigure
     [http://www.pragma-ade.com/show-gra.pdf]
     [page=1,width=\textwidth]}
```

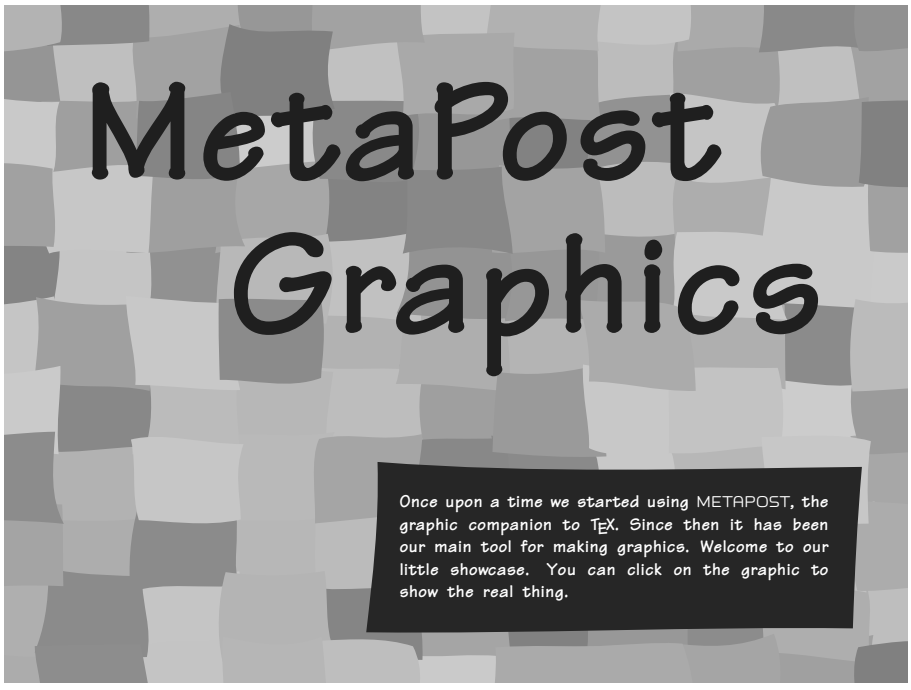


Figure 4. http handling

Other protocols, like FTP are also supported, so one can say:

```
\typefile {ftp://anonymous:@ctan.org/tex-archive/systems\
    /knuth/lib/plain.tex}
```

On the agenda is playing with database, but by the time that we enter that stage linking the `curl` libraries into LuaTEX should have taken place.

verbatim

The advance of LuaTEX also permitted us to play with a long standing wish of catcode tables, a mechanism to quickly switch between different ways of treating input characters. An example of a place where such changes take place is verbatim (and in ConTEXt also when dealing with XML input).

We already had encountered the phenomena that when piping back results from Lua to TEX, we needed to take care of catcodes so that TEX would see the input as we wished. Earlier experiments with applying `\scantokens` to a result and thereby interpreting the result conforming the current catcode regime was not sufficient or at least not handy enough, especially in the perspective of fully expandable Lua results. To be honest, `\scantokens` was rather useless for this purposes due to its pseudo file nature and its end–of–file handling but in LuaTEX we now have a convenient `\scantextokens` which has no side effects.

Once catcode tables were in place, and the relevant ConTEXt code adapted, I could start playing with one of the trickier parts of TEX programming: typesetting TEX using TEX, or verbatim. Because in ConTEXt verbatim is also related to buffering and pretty printing, all these mechanism were handled at once. It proved to be a pretty good testcase for writing Lua results back to TEX, because anything you can imagine can and will interfere (line endings, catcode changes, looking ahead for arguments, etc). This is one of the areas where Mark IV code will make things look more clean and understandable, especially because we could move all kind of postprocessing (needed for pretty printing, i.e. syntax highlighting) to Lua. Interesting is that the resulting code is not beforehand faster.

Pretty printing 1000 small (one line) buffers and 5000 simple `\type` commands perform as follows:

	TEX normal	TEX pretty	Lua normal	Lua pretty
buffer	2.5 (2.35)	4.5 (3.05)	2.2 (1.8)	2.5 (2.0)
inline	7.7 (4.90)	11.5 (7.25)	9.1 (6.3)	10.9 (7.5)

Between braces the runtime on Taco's more modern machine is shown. It's not that easy to draw conclusions from this because TEX uses files for buffers and with LUA we store buffers in memory. For inline verbatim, LUA call's bring some overhead, but with more complex content, this becomes less noticable. Also, the LUA code is probably less optimized than the TEX code, and we don't know yet what benefits a Just In Time LUA compiler will bring.

xml

Interesting is that the first experiments with XML processing don't show the expected gain in speed. This is due to the fact that the ConTEXt XML parser is highly optimized. However, if we want to load a whole XML file, for instance the formal ConTEXt interface specification `cont-en.xml`, then we can bring down loading time (as well as TEX memory usage) down from multiple seconds to a blink of the eyes. Experiments with internal mappings and manipulations demonstrated that we may not so much need an alternative for the current parser, but can add additional, special purpose ones.

We may consider linking XSLTPROC into LUATEX, but this is yet undecided. After all, the problem of typesetting does not really change, so we may as well keep the process of manipulating and typesetting separated.

multipass data

Those who know ConTEXt a bit will know that it may need multiple passes to typeset a document. ConTEXt not only keeps track of index entries, list entries, cross references, but also optimizes some of the output based on information gathered in previous passes. Especially so called two–pass data and positional information puts some demands on memory and runtime. Two–pass data is collapsed in lists because otherwise we would run out of memory (at least this was true years ago when these mechanisms were introduced). Positional information is stored in hashes and has always put a bit of a burden on the size of a so called utility file (ConTEXt stores all information in one auxiliary file).

These two datatypes were the first we moved to a LUA auxiliary file and eventually all information will move there. The advantage is that we can use efficient hashes (without limitations) and only need to run over the file once. And LUA is incredibly fast in loading the tables where we keep track of these things. For instance, a test file storing and reading 10.000 complex positions takes 3.2 seconds runtime with LUATEX but 8.7 seconds with traditional PDFTEX. Imagine what this will save when dealing with huge files (400 page 300 Meg files) that need three or more passes to be typeset. And, now we can without problems bump position tracking to milions of positions.

Initialization revised

Initializing LUATEX in such a way that it does what you want it to do your way can be tricky. This has to do with the fact that if we want to overload certain features (using callbacks) we need to do that before the orginals start doing their work. For instance, if we want to install our own file handling, we must make sure that the built–in file searching does not get initialized. This is particularly important when the built in search engine is based on the KPSE library. In that case the first serious file access will result in loading the `ls-R` filename databases, which will take an amount of time more or less linear with the size of the TEX trees. Among the

reasons why we want to replace KPSE are the facts that we want to access ZIP files, do more specific file searches, use HTTP, FTP and whatever comes around, integrate ConTEXt specific methods, etc.

Although modern operating systems will cache files in memory, creating the internal data structures (hashes) from the rather dumb files take some time. On the machine where I was developing the first experimental LUATEX code, we're talking about 0.3 seconds for PDFTEX. One would expect a LUA based alternative to be slower, but it is not. This may be due to the different implementation, but for sure the more efficient file cache plays a role as well. So, by completely disabling KPSE, we can have more advanced IO related features (like reading from ZIP files) at about the same speed (or even faster). In due time we will also support progname (and format) specific caches, which speeds up loading. In case one wonders why we bother about a mere few hundreds of milliseconds: imagine frequent runs from an editor or sub–runs during a job. In such situation every speed up matters.

So, back to initialization: how do we initialize LUATEX. The method described here is developed for ConTEXt but is not limited to this macro package; when one tells TEXEXEC to generate formats using the `--luatex` directive, it will generate the ConTEXt formats as well as MPTOPDF using this engine.

For practical reasons, the Lua based IO handler is KPSE compliant. This means that the normal `texmf.cnf` and `ls-R` files can be used. However, their content is converted in a more LUA friendly way. Although this can be done at runtime, it makes more sense to to this in advance using LUATOOLS. The files involved are:

input	**raw input**	**runtime input**	**runtime fallback**
	`ls-R`	`files.luc`	`files.lua`
`texmf.lua`	`temxf.cnf`	`configuration.luc`	`configuration.lua`

In due time LUATOOLS will generate the directory listing itself (for this some extra libraries need to be linked in). The configuration file(s) eventually will move to a LUA table format, and when a `texmf.lua` file is present, that one will be used.

```
luatools --generate
```

This command will generate the relevant databases. Optionally you can provide `--minimize` which will generate a leaner database, which in turn will bring down loading time to (on my machine) about 0.1 sec instead of 0.2 seconds. The `--sort` option will give nicer intermediate (`.lua`) files that are more handy for debugging.

When done, you can use LUATOOLS roughly in the same manner as KPSEWHICH, for instance to locate files:

```
luatools texnansi-lmr10.tfm
luatools --all tufte.tex
```

You can also inspect its internal state, for instance with:

```
luatools --variables  --pattern=TEXMF
luatools --expansions --pattern=context
```

This will show you the (expanded) variables from the configuration files. Normally you don't need to go that deep into the belly.

The LUATOOLS script can also generate a format and run LUATEX. For ConTEXt this is normally done with the TEXEXEC wrapper, for instance:

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

When dealing with this process we need to keep several things in mind:

- LUATEX needs a LUA startup file in both ini and runtime mode
- these files may be the same but may also be different

- here we use the same files but a compiled one in runtime mode
- we cannot yet use a file location mechanism

A `.luc` file is a precompiled LUA chunk. In order to guard consistency between LUA code and tex code, ConTEXt will preload all LUA code and store them in the bytecode table provided by LUATEX. How this is done, is another story. Contrary to these tables, the initialization code can not be put into the format, if only because at that stage we still need to set up memory and other parameters.

In our case, especially because we want to overload the IO handler, we want to store the startup file in the same path as the format file. This means that scripts that deal with format generation also need to take care of (relocating) the startup file. Normally we will use TEXEXEC but we can also use LUATOOLS.

Say that we want to make a plain format. We can call LUATOOLS as follows:

```
luatools --ini plain
```

This will give us (in the current path):

```
120,808 plain.fmt
  2,650 plain.log
 80,767 plain.lua
 64,807 plain.luc
```

From now on, only the `plain.fmt` and `plain.luc` file are important. Processing a file

```
test \end
```

can be done with:

```
luatools --fmt=./plain.fmt test
```

This returns:

```
This is luaTeX, Version 3.141592-0.1-alpha-20061018 (Web2C 7.5.5)
(./test.tex [1] )
Output written on test.dvi (1 page, 260 bytes).
Transcript written on test.log.
```

which looks rather familiar. Keep in mind that at this stage we still run good old Plain TEX. In due time we will provide a few files that will making work with LUA more convenient in Plain TEX, but at this moment you can already use for instance `\directlua`.

In case you wonder how this is related to ConTEXt, well only to the extend that it uses a couple of rather generic ConTEXt related LUA files.

ConTEXt users can best use TEXEXEC which will relocate the format related files to the regular engine path. In LUATOOLS terms we have two choices:

```
luatools --ini cont-en
luatools --ini --compile cont-en
```

The difference is that in the first case `context.lua` is used as startup file. This LUA file creates the `cont-en.luc` runtime file. In the second call LUATOOLS will create a `cont-en.lua` file and compile that one. An even more specific call would be:

```
luatools --ini --compile --luafile=blabla.lua          cont-en
luatools --ini --compile --lualibs=bla-1.lua,bla-2.lua cont-en
```

This call does not make much sense for ConTEXt. Keep in mind that LUATOOLS does not set up user specific configurations, for instance the `--all` switch in TEXEXEC will set up all patterns.

I know that it sounds a bit messy, but till we have a more clear picture of where LuaTeX is heading this is the way to proceed. The average ConTeXt user won't notice those details, because TeXexec will take care of things.

Currently we follow the TDS and web2c conventions, but in the future we may follow different or additional approaches. This may as well be driven by more complex IO models. For the moment extensions still fit in. For instance, in order to support access to remote resources and related caching, we have added to the configuration file the variable:

```
TEXMFCACHE = $TMP;$TEMP;$TMPDIR;$HOME;$TEXMFVAR;$VARTEXMF;.
```

An example: CalcMath

introduction

For a long time TeX's way of coding math has dominated the typesetting world. However, this kind of coding is not that well suited for non academics, like schoolkids. Often kids do know how to key in math because they use advanced calculators. So, when a couple of years ago we were implementing a workflow where kids could fill in their math workbooks (with exercises) on–line, it made sense to support so called Texas Instruments math input. Because we had to parse the form data anyway, we could use a `[[` and `]]` as math delimiters instead of `$`. The conversion too place right after the form was received by the web server.

`sin(x) + x^2 + x^(1+x) + 1/x^2`	$\sin(x) + x^2 + x^{1+x} + \frac{1}{x^2}$
`mean(x+mean(y))`	$\overline{x + \overline{y}}$
`int(a,b,c)`	$\int_b^a c$
`(1+x)/(1+x) + (1+x)/(1+(1+x)/(1+x))`	$\frac{1+x}{1+x} + \frac{1+x}{1+\frac{1+x}{1+x}}$
`10E-2`	10×10^{-2}
`(1+x)/x`	$\frac{1+x}{x}$
`(1+x)/12`	$\frac{1+x}{12}$
`(1+x)/-12`	$\frac{1+x}{-12}$
`1/-12`	$\frac{1}{-12}$
`12x/(1+x)`	$\frac{12x}{1+x}$
`exp(x+exp(x+1))`	$e^{x+e^{x+1}}$
`abs(x+abs(x+1)) + pi + inf`	$\lvert x + \lvert x + 1\rvert\rvert + \pi + \mathrm{inf}$
`Dx Dy`	$\frac{dx}{dx}\frac{dy}{dx}$
`D(x+D(y))`	$\frac{d}{dx}(x + \frac{d}{dx}(y))$
`Df(x)`	$f'(x)$
`g(x)`	$g(x)$
`sqrt(sin^2(x)+cos^2(x))`	$\sqrt{\sin^2(x) + \cos^2(x)}$

By combining Lua with TeX, we can do the conversion from calculator math to TeX immediately, without auxiliary programs or complex parsing using TeX macros.

tex
In a ConTEXt source one can use the `\calcmath` command, as in:

```
The strange formula \calcmath {sqrt(sin^2(x)+cos^2(x))} boils
down to ...
```

One needs to load the module first, using:

```
\usemodule[calcmath]
```

Because the amount of code involved is rather small, eventually we may decide to add this support to the Mark IV kernel.

xml
Coding math in TEX is rather efficient. In XML one needs way more code. Presentation MATHML provides a few basic constructs and boils down to combining those building blocks. Content MATHML is better, especially from the perspective of applications that need to do interpret the formulas. It permits for instance the ConTEXt content MATHML handler to adapt the rendering to cultural driven needs. The OPENMATH way of coding is like content MATHML, but more verbose with less tags. Calculator math is more restrictive than TEX math and less verbose than any of the XML variants. It looks like:

```
<icm>sqrt(sin^2(x)+cos^2(x))</icm> test
```

And in display mode:

```
<dcm>sqrt(sin^2(x)+cos^2(x))</dcm> test
```

speed
This script (which you can find in the ConTEXt distribution as soon as the Mark IV code variants are added) is the first real TEX related LUA code that I wrote; so far I had only written some wrapping and spell checking code for the SCITE editor. It also made a nice demo for a couple of talks that I held at usergroup meetings. The script has a lot of expressions. These convert one string into another. They are less powerful than regular expressions, but pretty fast and adequate. The feature I miss most is alternation like `(l|st)uck` but it's a small price to pay. As the LUA manual explains: adding a POSIX compliant regexp parser would take more lines of code than LUA currently does.

On my machine, running this first version took 3.5 seconds for 2500 times typesetting the previously shown square root of sine and cosine. Of this, 2.1 seconds were spent on typesetting and 1.4 seconds on converting. After optimizing the code, 0.8 seconds were used for conversion. A stand alone LUA takes .65 seconds, which includes loading the interpreter. On a test of 25.000 sample conversions, we could gain some 20% conversion time using the LUAJIT just in time compiler.

Hans Hagen

Display Math in ConTEXt

ConTEXt rehab for amsmath addicts

Abstract
This article explains how to do various kinds of alignments in ConTEXt. A visual output is presented, and it is then shown how that effect can be achieved in LaTEX and ConTEXt. We hope that article will make the transition from LaTEX with `amsmath` package to ConTEXt easier.

Keywords
ConTEXt, LaTEX, math alignment, amsmath

Introduction

Plain TEX provides several macros like `\eqalign`, `\eqalignno`, `\displaylines`, `\matrix`, `\pmatrix`, `\cases`, and `\halign`, for math alignments. These macros are adequate for most constructions that occur in practice. AMS-TeX and the `amsmath` package for LaTEX supply math alignment environments that provide a layer of abstraction for the user and makes it (slightly) easier for him/her to type the common math alignments. Most people learning TEX these days start from LaTEX and those writing substantial math use the `amsmath` package; they know nothing about the plain TEX math alignment macros. In earlier versions of ConTEXt, since the plain TEX macros could be used, no additional macros for math alignments were provided. This made writing math alignments difficult for users who came to ConTEXt with a LaTEX background. They did not know about plain TEX macros and kept looking for something equivalent to the `amsmath` package. There was an `amsl` package module by Giuseppe Bilotta, but it was very limited. Moreover, doing alignments with multiple alignment points in plain TEX requires a good understanding of the TEX alignment mechanism; making them obscure for a typical user. This resulted in a general impression that ConTEXt does not handle math very well.

Recently (in January 2006 to be precise), Hans added math alignment macros in ConTEXt. These macros provide a very nice user interface to plain TEX's alignment mechanism; they can be used to achieve the functionality of `amsmath` package macros; and, like all user macros in ConTEXt, they are easy to customize. These macros, however, neither copy the user interface of `amsmath` package, nor the implementation. So, translating your existing LaTEX math code into ConTEXt requires some effort and the result is not necessarily, pixel by pixel, identical.

In this article, I describe how to convert the common alignment constructions from LaTEX to ConTEXt, highlighting some of the flexibility offered by ConTEXt. This is a *visual* document: I first show how the output should look like, then present LaTEX and ConTEXt examples that give that output. This article is not meant as a tutorial for math alignments in LaTEX or ConTEXt, and I do not explain the LaTEX and the ConTEXt syntax. The article is not exhaustive; it provides a small sample of math alignments that can be done using LaTEX and ConTEXt. For an indepth treatment of LaTEX's math capabilities see Herbert Voß's mathmode.[1] For a introduction to ConTEXt math alignment see My Way on `\startalign` and friends.[2] The objective

of this article is not to compare the features of these two macro packages, rather it is to show that ConTEXt is capable of handling "complicated" math alignments.

Math Alignments

ConTEXt provides `mathalignment` series of macros (`\definemathalignment`, `\setupmathalignment`, `\startmathalignment`, and `\stopmathalignment`) to take care of the different math alignments. Below, I describe some common math constructs, and examples of how to achieve them in LaTEX and ConTEXt.

gather
The `gather` environment of `amsmath` package allows you to write multi–line formulas with each line center aligned. It is perhaps the simplest form of "alignment". In ConTEXt the same effect can be achieved using appropriate options to `\startmathalignment`.

$$\blacksquare\blacksquare = \blacksquare\blacksquare + \blacksquare\blacksquare\blacksquare + \blacksquare\blacksquare + \blacksquare$$
$$\blacksquare\blacksquare = \blacksquare\blacksquare + \blacksquare\blacksquare\blacksquare + \blacksquare\blacksquare + \blacksquare\blacksquare + \blacksquare\blacksquare\blacksquare$$

In LaTEX

```
\begin{gather}
  v = u  + at,           \\
  d = ut + \frac12 at^2.
\end{gather}
```

$$v = u + at, \tag{1}$$
$$d = ut + \frac{1}{2}at^2. \tag{2}$$

In ConTEXt

```
\placeformula \startformula
  \startmathalignment[n=1]
    \NC v = u  + at,           \NR[+]
    \NC d = ut + \frac12 at^2. \NR[+]
  \stopmathalignment
\stopformula
```

$$v = u + at, \tag{1}$$
$$d = ut + \frac{1}{2}at^2. \tag{2}$$

left gather
Sometimes one wants multi–line formulas, where each line is left or right aligned, rather than center aligned as in the `gather` environment. Although, LaTEX does not provide any in–built environment for such constructions, it is easy to exploit the `align` environment to achieve this output. In ConTEXt passing `align=left` to `\startmathalignment` gives the desired output.

$$\blacksquare\blacksquare\blacksquare = \blacksquare\blacksquare + \blacksquare$$
$$\blacksquare\blacksquare\blacksquare\blacksquare = \blacksquare\blacksquare + \blacksquare\blacksquare + \blacksquare\blacksquare + \blacksquare + \blacksquare\blacksquare + \blacksquare\blacksquare$$

In LaTeX

```
\begin{align}
  & v = u  + at,          \\
  & d = ut + \frac12 at^2.
\end{align}
```

$$v = u + at, \tag{1}$$

$$d = ut + \frac{1}{2}at^2. \tag{2}$$

In ConTeXt

```
\placeformula \startformula
  \startmathalignment[n=1,align=left] %align=left does the magic
    \NC v = u  + at,           \NR[+]
    \NC d = ut + \frac12 at^2. \NR[+]
  \stopmathalignment
\stopformula
```

$$v = u + at, \tag{1}$$

$$d = ut + \frac{1}{2}at^2. \tag{2}$$

right gather
For multi–line formulas with each line right aligned, in LaTeX you can exploit the `align` environment, while in ConTeXt you need to pass `align=right` to `\startmathalignment`

$$\blacksquare\blacksquare = \blacksquare\blacksquare + \blacksquare\blacksquare + \blacksquare\blacksquare\blacksquare + \blacksquare\blacksquare$$

$$\blacksquare\blacksquare = \blacksquare\blacksquare + \blacksquare\blacksquare$$

In LaTeX

```
\begin{align}
  v = u  + at ,          & \\
  d = ut + \frac12 at^2.  &
\end{align}
```

$$v = u + at, \tag{1}$$

$$d = ut + \frac{1}{2}at^2. \tag{2}$$

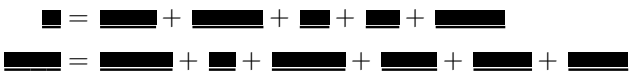
In ConTeXt

```
\placeformula \startformula
  \startmathalignment[n=1,align=right] %align=right does the magic
    \NC v = u  + at,           \NR[+]
    \NC d = ut + \frac12 at^2. \NR[+]
  \stopmathalignment
\stopformula
```

$$v = u + at, \tag{1}$$

$$d = ut + \frac{1}{2}at^2. \tag{2}$$

align
This is the simplest and the most widely used form of alignment. In the simplest case, there are two columns, one right aligned and the other left aligned. In LaTeX the `align` environment takes care of such alignments; in ConTeXt `\startmathalignment`.



In LaTEX

```
\begin{align}
  v &= u  + at,           \\
  d &= ut + \frac12 at^2.
\end{align}
```

$$v = u + at, \tag{1}$$

$$d = ut + \frac{1}{2}at^2. \tag{2}$$

In ConTEXt

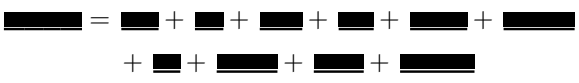
```
\placeformula \startformula
  \startmathalignment
    \NC v \NC = u  + at,           \NR[+]
    \NC d \NC = ut + \frac12 at^2. \NR[+]
  \stopmathalignment
\stopformula
```

$$v = u + at, \tag{1}$$

$$d = ut + \frac{1}{2}at^2. \tag{2}$$

split

The `split` environment of `amsmath` package is used for writing a single formula which needs more than one line. The whole formula gets a single number. In ConTEXt you have to manually specify which line to number.



In LaTEX

```
\begin{equation} \begin{split}
  (x+1)^8 ={} & x^8 + 8 x^7 + 28 x^6 + 56 x^5 + 70 x^4 \\
              & + 56 x^3 + 28 x^2 + 8 x + 1.
\end{split} \end{equation}
```

$$\begin{aligned}(x+1)^8 = {} & x^8 + 8x^7 + 28x^6 + 56x^5 + 70x^4 \\ & + 56x^3 + 28x^2 + 8x + 1.\end{aligned} \tag{1}$$

In ConTEXt

```
\placeformula \startformula
  \startmathalignment
  \NC (x+1)^8 = \NC x^8 + 8 x^7 + 28 x^6 + 56 x^5 + 70 x^4 \NR
  \NC           \NC + 56 x^3 + 28 x^2 + 8 x + 1.           \NR[+]
  \stopmathalignment
\stopformula
```

$$\begin{aligned}(x+1)^8 = {} & x^8 + 8x^7 + 28x^6 + 56x^5 + 70x^4 \\ & + 56x^3 + 28x^2 + 8x + 1.\end{aligned} \tag{1}$$

Notice that in LaTeX the spacing around $=$ in the first line has to be manually corrected by typing `{}`. ConTeXt takes care of this automatically. In LaTeX you can control the position of the tag with `tbtags` or `centertags` option to the `amsmath` package. Right now, with the math alignment macros in ConTeXt you can only achieve the result equivalent to `tbtags`. To get the result equivalent to `centertags` you have to use the `\eqalign` macro of plain TeX.

alignat

The `alignat` environment of `amsmath` package allows you to align at several places. The alignment order alternates between right and left aligned columns. In ConTeXt the same effect can be achieved by `m=...` option of `\startmathalignment`.

In LaTeX

```
\begin{alignat}{2}
  \nabla\cdot  \mathbf E &=  \frac{\rho}{\varepsilon_0},  \qquad
& \nabla\times \mathbf E &= -\frac{\partial \mathbf B}{\partial t},\\
  \nabla\cdot  \mathbf B &= 0,
& \nabla\times \mathbf B &= \mu_0{\mathbf j}+\varepsilon_0\mu_0
                            \frac{\partial \mathbf E}{\partial t}.
\end{alignat}
```

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0}, \qquad \nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}, \tag{1}$$

$$\nabla \cdot \mathbf{B} = 0, \qquad \nabla \times \mathbf{B} = \mu_0 \mathbf{j} + \varepsilon_0 \mu_0 \frac{\partial \mathbf{E}}{\partial t}. \tag{2}$$

In ConTeXt

```
\placeformula \startformula
  \startmathalignment[m=2,distance=2em]%Notice distance=2em
  \NC \nabla\cdot  \bf E \NC=  \frac{\rho}{\varepsilon_0},
  \NC \nabla\times \bf E \NC= -\frac{\partial \bf B}{\partial t},\NR[+]
  \NC \nabla\cdot  \bf B \NC= 0,
  \NC \nabla\times \bf B \NC= \mu_0{\bf j}+\varepsilon_0\mu_0
                              \frac{\partial \bf E}{\partial t}. \NR[+]
 \stopmathalignment
\stopformula
```

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0}, \qquad \nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}, \tag{1}$$

$$\nabla \cdot \mathbf{B} = 0, \qquad \nabla \times \mathbf{B} = \mu_0 \mathbf{j} + \varepsilon_0 \mu_0 \frac{\partial \mathbf{E}}{\partial t}. \tag{2}$$

flalign

The `flalign` environment is the same as `alignat` environment but with the equations a little more "out spaced". In ConTeXt you can control the space between the "blocks" by `distance=...` option to `\startmathalignment`.

In LaTEX

```
\begin{flalign*}
  \nabla\cdot  \mathbf E &=  \frac{\rho}{\varepsilon_0},
& \nabla\times \mathbf E &= -\frac{\partial \mathbf B}{\partial t}.\\
  \nabla\cdot  \mathbf B &= 0,
& \nabla\times \mathbf B &= \mu_0{\mathbf j}+\varepsilon_0\mu_0
                            \frac{\partial \mathbf E}{\partial t}.
\end{flalign*}
```

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0}, \qquad \nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}.$$

$$\nabla \cdot \mathbf{B} = 0, \qquad \nabla \times \mathbf{B} = \mu_0 \mathbf{j} + \varepsilon_0 \mu_0 \frac{\partial \mathbf{E}}{\partial t}.$$

In ConTEXt

```
\startformula
  \startmathalignment[m=2,distance=2em plus 1 fil]%Notice distance=...
   \NC \nabla\cdot  \bf E \NC=  \frac{\rho}{\varepsilon_0}, \NR
   \NC \nabla\times \bf E \NC= -\frac{\partial \bf B}{\partial t}, \NR
   \NC \nabla\cdot  \bf B \NC= 0,
   \NC \nabla\times \bf B \NC= \mu_0{\bf j}+\varepsilon_0\mu_0
                               \frac{\partial \bf E}{\partial t}.  \NR
 \stopmathalignment
\stopformula
```

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0}, \qquad \nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t},$$

$$\nabla \cdot \mathbf{B} = 0, \qquad \nabla \times \mathbf{B} = \mu_0 \mathbf{j} + \varepsilon_0 \mu_0 \frac{\partial \mathbf{E}}{\partial t}.$$

intertext
The `\intertext` macro from `amsmath` allows you to break the alignment and write some text, which does not affect the alignment. ConTEXt provides the `\intertext` macro and a `\startintertext`, `\stopintertext` environment for the same.

$$\blacksquare\blacksquare = \blacksquare + \blacksquare + \blacksquare\blacksquare + \blacksquare$$

$$\blacksquare\blacksquare \quad \blacksquare\blacksquare \quad \blacksquare\blacksquare \quad \blacksquare\blacksquare \quad \blacksquare\blacksquare$$

$$\blacksquare = \blacksquare + \blacksquare + \blacksquare\blacksquare + \blacksquare + \blacksquare\blacksquare + \blacksquare$$

In LaTEX

```
\begin{align*}
  \cos 2\theta &= \cos^2 \theta + \sin^2 \theta \\
  \intertext{replace $\sin^2 \theta$ by $1 - \cos^2 \theta$}
  &= 2\cos^2 \theta - 1
\end{align*}
```

$$\cos 2\theta = \cos^2\theta + \sin^2\theta$$

replace $\sin^2\theta$ by $1 - \cos^2\theta$

$$= 2\cos^2\theta - 1$$

In ConTEXt

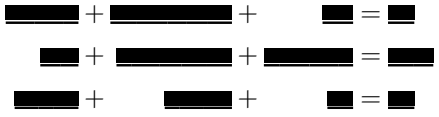
```
\startformula
  \startmathalignment
    \NC \cos 2\theta \NC= \cos^2 \theta + \sin^2 \theta \NR
    \intertext{replace $\sin^2 \theta$ by $1 - \cos^2 \theta$}
    \NC \NC = 2\cos^2 \theta - 1 \NR
  \stopmathalignment
\stopformula
```

$$\cos 2\theta = \cos^2\theta + \sin^2\theta$$

replace $\sin^2\theta$ by $1 - \cos^2\theta$

$$= 2\cos^2\theta - 1$$

linear equations
In LaTeX linear equations can be handled using `alignat` environment; in ConTeXt appropriate options to `\startmathalignment` take care of this construction.[3]



In LaTeX

```
\begin{alignat}{5}
    x_1  & {} + {}& x_2     &{} + {}& 6x_3  &{} = {}& 170, \\
   3x_1  & {} - {}& 110x_2  &{} - {}& x_3   &{} = {}& 4,   \\
  14x_1  & {} + {}& 13x_2   &{} + {}& 10x_3 &{} = {}& 25.
\end{alignat}
```

$$\begin{aligned} x_1 + \quad x_2 + \; 6x_3 &= 170, && (1)\\ 3x_1 - 110x_2 - \quad x_3 &= \quad 4, && (2)\\ 14x_1 + \; 13x_2 + 10x_3 &= \; 25. && (3) \end{aligned}$$

In ConTEXt

```
\placeformula \startformula
  \startmathalignment
                [n=7,align={right,left,right,left,right,left,right}]
    \NC   x_1  \NC + \NC x_2     \NC + \NC 6x_3  \NC = \NC 170, \NR[+]
    \NC  3x_1  \NC - \NC 110x_2  \NC - \NC x_3   \NC = \NC 4,   \NR[+]
    \NC 14x_1  \NC + \NC 13x_2   \NC + \NC 10x_3 \NC = \NC 25.  \NR[+]
  \stopmathalignment
\stopformula
```

$$x_1 + x_2 + 6x_3 = 170, \tag{1}$$
$$3x_1 - 110x_2 - x_3 = 4, \tag{2}$$
$$14x_1 + 13x_2 + 10x_3 = 25. \tag{3}$$

In LaTEX we are limited to left and right aligned columns. In ConTEXt it is easy to change the alignment of individual columns. For example

```
\placeformula \startformula
  \startmathalignment[n=7,
           align={middle,middle,middle,middle,middle,middle,middle}]
    \NC   x_1  \NC + \NC x_2     \NC + \NC 6x_3  \NC = \NC 170, \NR[+]
    \NC  3x_1  \NC - \NC 110x_2  \NC - \NC x_3   \NC = \NC 4,   \NR[+]
    \NC 14x_1  \NC + \NC 13x_2   \NC + \NC 10x_3 \NC = \NC 25.  \NR[+]
  \stopmathalignment
\stopformula
```

$$x_1 + x_2 + 6x_3 = 170, \tag{1}$$
$$3x_1 - 110x_2 - x_3 = 4, \tag{2}$$
$$14x_1 + 13x_2 + 10x_3 = 25. \tag{3}$$

multi–column numbered equations
Sometimes, while writing formulas in blocks, you need to number formulas in all blocks. I do not know of any easy way to do this in LaTEX. Herbert Voß's Mathmode[1] has an example in Section 73 of using `tabular` to achieve this effect. ConTEXt provides `\startformulas` for multi–column formulas, which allows numbering of formulas in each column.

$$\blacksquare = \blacksquare + \blacksquare + \blacksquare \tag{1}$$
$$\blacksquare = \blacksquare + \blacksquare + \blacksquare$$

$$\blacksquare = \blacksquare$$
$$\blacksquare = \blacksquare \tag{2}$$

```
\placeformula \startformulas
 \startformula \startmathalignment
  \NC \nabla\cdot  \bf E \NC=  \frac{\rho}{\varepsilon_0}, \NR[+]
  \NC \nabla\cdot  \bf B \NC= 0,                          \NR[+]
 \stopmathalignment \stopformula
 \startformula \startmathalignment
  \NC \nabla\times \bf E \NC= -\frac{\partial \bf B}{\partial t},\NR[+]
  \NC \nabla\times \bf B \NC= \mu_0{\bf j}+\varepsilon_0\mu_0
                             \frac{\partial \bf E}{\partial t}. \NR[+]
 \stopmathalignment \stopformula
\stopformulas
```

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0}, \qquad (1) \qquad \nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}, \qquad (3)$$

$$\nabla \cdot \mathbf{B} = 0, \qquad (2) \qquad \nabla \times \mathbf{B} = \mu_0 \mathbf{j} + \varepsilon_0 \mu_0 \frac{\partial \mathbf{E}}{\partial t}. \qquad (4)$$

Defining your own alignments
In the examples given above, I passed the arguments to `\startmathalignment`. This can be repetitive if you have to use the same alignment many times. ConTEXt provides `\definemathalignment` to define a new math alignments. Suppose you have to type a lot of linear equations, you can define your own alignment as follows

```
\definemathalignment
  [linearequations]
  [n=7,align={middle,middle,middle,middle,middle,middle,middle}]

\placeformula \startformula
  \startlinearequations
    \NC   x_1  \NC + \NC x_2     \NC + \NC 6x_3  \NC = \NC 170, \NR[+]
    \NC  3x_1  \NC - \NC 110x_2  \NC - \NC x_3   \NC = \NC 4,   \NR[+]
    \NC 14x_1  \NC + \NC 13x_2   \NC + \NC 10x_3 \NC = \NC 25.  \NR[+]
  \stoplinearequations
\stopformula
```

$$\begin{aligned} x_1 + x_2 + 6x_3 &= 170, && (1)\\ 3x_1 - 110x_2 - x_3 &= 4, && (2)\\ 14x_1 + 13x_2 + 10x_3 &= 25. && (3) \end{aligned}$$

You can define similar alignments for each special case that you have to use.

Matrix and Arrays

ConTEXt provides `mathmatrix` series of macros (`\definemathmatrix`, `\setupmathmatrix`, `\startmathmatrix`, and `\stopmathmatrix`) to take care of matrix alignments. These macros can provide functionality of `array` environment as well as the `matrix` series of macros from `amsmath` package.

Simple Matrix
A matrix is a collection of objects that are arranged in rows and columns. In LaTEX this alignment is provided by the `array` environment. In ConTEXt `\startmathmatrix` provides this feature.

In LaTEX

```
\begin{equation*}
\setlength{\arraycolsep}{1em}
  \begin{array}{ccc}
    A   &   B & C  \\
    AA  &  BB & CC \\
    AAA & BBB & CCC
  \end{array}
\end{equation*}
```

$$\begin{matrix} A & B & C \\ AA & BB & CC \\ AAA & BBB & CCC \end{matrix}$$

In ConTEXt

```
\startformula
  \startmathmatrix[n=3]
    \NC A   \NC B   \NC C   \NR
    \NC AA  \NC BB  \NC CC  \NR
    \NC AAA \NC BBB \NC CCC \NR
  \stopmathmatrix
\stopformula
```

$$\begin{matrix} A & B & C \\ AA & BB & CC \\ AAA & BBB & CCC \end{matrix}$$

In LaTEX the alignment of each column can be changed by the `r,c,l` options to `array`. In ConTEXt you need to pass appropriate arguments to `align=...`.
In LaTEX

```
\begin{equation*}
\setlength{\arraycolsep}{1em}
  \begin{array}{lcr}
    A   &   B & C  \\
    AA  &  BB & CC \\
    AAA & BBB & CCC
  \end{array}
\end{equation*}
```

$$\begin{array}{lcr} A & B & C \\ AA & BB & CC \\ AAA & BBB & CCC \end{array}$$

In ConTEXt

```
\startformula
  \startmathmatrix[n=3,align={left,middle,right}]
    \NC A   \NC B   \NC C   \NR
    \NC AA  \NC BB  \NC CC  \NR
    \NC AAA \NC BBB \NC CCC \NR
  \stopmathmatrix
\stopformula
```

$$\begin{matrix} A & B & C \\ AA & BB & CC \\ AAA & BBB & CCC \end{matrix}$$

pmatrix, et. al

The `amsmath` package provides `pmatrix`, `bmatrix`, etc. environments that make it easy to typeset matrix surrounded by delimiters. In ConTEXt it is straightforward to define such matrices uses `\definemathmatrix`
In LaTEX

```
\begin{equation*}
  A = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}
\end{equation*}
```

$$A = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

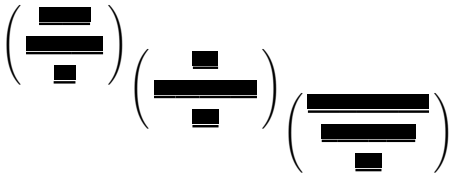
In ConTEXt

```
\definemathmatrix
  [pmatrix]
  [left={\left(\,},right={\,\right)}]

\startformula
  A = \startpmatrix 1 \NR 2 \NR 3 \NR \stoppmatrix
\stopformula
```

$$A = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

delarray package



The `delarray` package in LaTeX allows you to typeset arrays with properly scaled delimiters, even when the array is not middle aligned to the baseline. In ConTEXt the `\startmathmatrix` takes care of proper scaling of delimiters.
In LaTeX

```
\begin{equation*}
  \begin{array}[b]({c}) 1 \\ 2 \\ 3 \end{array}
  \begin{array}[c]({c}) 1 \\ 2 \\ 3 \end{array}
  \begin{array}[t]({c}) 1 \\ 2 \\ 3 \end{array}
\end{equation*}
```

$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

In ConTEXt

```
\definemathmatrix
  [pmatrix]
  [left={\left(\,},right={\,\right)}]

\startformula
  \startpmatrix[location=low]    1 \NR 2 \NR 3 \NR \stoppmatrix
  \startpmatrix[location=middle] 1 \NR 2 \NR 3 \NR \stoppmatrix
  \startpmatrix[location=high]   1 \NR 2 \NR 3 \NR \stoppmatrix
\stopformula
```

$$\binom{1}{2}_{\binom{1}{2}_{\binom{1}{2}}}$$

Cases

Cases is another common math alignment.

$$\blacksquare\blacksquare\blacksquare\blacksquare = \begin{cases} \blacksquare\blacksquare\blacksquare\blacksquare + \blacksquare\blacksquare\blacksquare\blacksquare, & \blacksquare\blacksquare \blacksquare\blacksquare\blacksquare\blacksquare\blacksquare \blacksquare\blacksquare \\ \blacksquare\blacksquare\blacksquare + \blacksquare\blacksquare\blacksquare + \blacksquare\blacksquare\blacksquare, & \blacksquare\blacksquare\blacksquare\blacksquare \blacksquare\blacksquare\blacksquare\blacksquare\blacksquare \end{cases}$$

The amsmath package provides a cases environment to build such alignments. ConTEXt provides \startmathcases.
In LaTEX

```
\begin{equation*}
  |x| =
  \begin{cases}
     x, & \text{if $x \ge 0$;} \\
    -x, & \text{otherwise.}
  \end{cases}
\end{equation*}
```

$$|x| = \begin{cases} x, & \text{if } x \geqslant 0; \\ -x, & \text{otherwise.} \end{cases}$$

In ConTEXt

```
\startformula
  |x| =
  \startmathcases
    \NC  x, \NC if $x \ge 0$ ; \NR
    \NC -x, \NC otherwise.     \NR
  \stopmathcases
\stopformula
```

$$|x| = \left\{ \begin{array}{ll} x, & \text{if } x \geqslant 0 ; \\ -x, & \text{otherwise.} \end{array} \right.$$

In the cases environment, the rows are set in textstyle. The mathtools package package provides a dcases environment to set the rows in displaystyle. In ConTEXt you can set the rows in displaystyle by passing style=\displaystyle to \startmathcases (or defining a new cases structure using \definemathcases).
In LaTEX

```
\begin{equation*}
  f(x) =
  \begin{dcases}
    \int_0^x g(y)\,dy,    & \text{if $x \ge 0$;} \\
    \int_{-x}^0 g(y)\,dy, & \text{otherwise.}
  \end{dcases}
\end{equation*}
```

$$f(x) = \begin{cases} \int_0^x g(y)\,dy, & \text{if } x \geqslant 0; \\ \int_{-x}^0 g(y)\,dy, & \text{otherwise.} \end{cases}$$

In ConTEXt

```
\startformula
  f(x) =
  \startmathcases[style=\displaystyle]
    \NC \int_0^x g(y)\,dy,    \NC if $x \ge 0$; \NR
    \NC \int_{-x}^0 g(y)\,dy, \NC otherwise.    \NR
  \stopmathcases
\stopformula
```

$$f(x) = \begin{cases} \int_0^x g(y)\,dy, & \text{if } x \geqslant 0; \\ \int_{-x}^0 g(y)\,dy, & \text{otherwise.} \end{cases}$$

Predefined Alignments

ConTEXt already has

```
\definemathalignment[align]
\definemathmatrix[matrix]
\definemantcases[cases]
```

defined. This means that in all the above examples, you can shorten `\startmathalignment ...\stopmathalignment` to `\startalign ...\stopalign`, `\startmathmatrix ...\stopmathmatrix` to `\startmatrix ...\stopmatrix`, and `\startmathcases ...\stopmathcases` to `\startalign ...\stopalign`.

Conclusion

ConTEXt now provides macros for math alignments. This makes it easier for the users to write complicated math alignments in ConTEXt. The syntax is consistent with the rest of ConTEXt macros, and thereby different from `amsmath` package syntax. Hopefully, this article will help eliminate the myth that ConTEXt is not able to handle complicated math. In ConTEXt features are added on user requests; so if there is something that you need which is not present in ConTEXt, ask for a feature request on the mailing list.

Notes

1. Herbert Voß, "*Math mode,*" available from http://tug.ctan.org/cgi-bin/getFile.py?fn=/info/math/voss/mathmode/Mathmode.pdf
2. Aditya Mahajan, "*My Way on* `\startalign` *and friends,*" available from http://dl.contextgarden.net/myway/mathalign.pdf
3. Compare these solutions from Exercise 22.9 in the TEXbook.

Aditya Mahajan
adityam@umich.edu

Metapost Developments

Abstract
The new release of metapost includes some new features as well as a number of bugfixes. The new functionality includes: the possibility to use a template for the naming of output files; support for cmyk and greyscale color models; per-object Postscript specials; the option to generate Encapsulated Postscript files adhering to Adobe's Document Structuring Conventions; the ability to embed re-encoded and/or subsetted fonts; and support for the GNU implementation of troff (groff).

Introduction

Version 0.901 of Metapost was released at BachoTEX 2005. It was mostly a bugfix release, that featured and updated manual and the new `mpversion` primitive on top of a set of bugfixes.

At that time, a new version was promised for the autumn. In hindsight, that was overly optimistic. It is now already the summer of 2006, and the feature set for version 1.0 is now finally frozen. It will be released in time for TEXLive 2006 and (hopefully MikTEX 2.5), so to an average user not much time will be lost by the delay.

Bugfixes

Stability issues

In previous versions of Metapost, the size of the memory array was not stored in mem file. But in Web2c–base systems, the memory sizes are dynamic and the size that should be used by the executable can change depending on the command–line invocation. This discrepancy resulted in a number of painful and unexpected bugs.

- Disappearing specials from the output
- Incorrect error messages
- Unexplained crashes

This problem will be tackled by storing the required minimum memory sizes in the memory dump file. If an unsolvable mismatch occurs, an error message will be issued.

turningnumber

The current (0.9) Metapost executable has a very simple algorithm to calculate the `turningnumber` operation. It simply connects the path's points using straight segments, adds up all the angles between those segments, and then divides the result by 360. This only works well if the path segments are well-behaved i.e. they do not self-intersect.

This is already an improvement over the old code in the sense that when it is wrong, it is predictably wrong. But it was a temporary measure, and the next version contains completely new code that calculates true curvature for the path segments.

The new algorithm is based on a mailing list discussion between members of the group. It will be slower, but (finally) 100% accurate.

New features

File-name templates

The first of the new feature is support for output filename templates. These templates use `printf`-style escape sequences and are re-evaluated before each shipout. Numeric fields can be left-padded to a user-supplied width by prepended zeroes.
The new primitive command is `filenametemplate`, and it is a string-valued command. The syntax is as simple as:

```
filenametemplate "%j-%3c.eps";
beginfig(1);
  draw p;
endfig;
```

If the file is saved as `test.mp`, then this will create the output file `test-001.eps` instead of `test.1` of previous versions.

A small set of escape sequences are possible, see table 1 for details.

To ensure compatibility with older files, the default value of `filenametemplate` is `%j.%c`. If you assign an empty string, it will revert to that default.

CMYK color model

Support will be added for the industry-standard CMYK color model. In the simples form this looks like:

%%	A percent sign
%j	The current jobname
%⟨0-9⟩c	The charcode value
%⟨0-9⟩y	The current year
%⟨0-9⟩m	The numeric month
%⟨0-9⟩d	The day of the month
%⟨0-9⟩H	The hour
%⟨0-9⟩M	The minute

Table 1. Allowed escape sequences for `filenametemplate`

```
beginfig(1);
   draw fullcircle
      withcmykcolor (1,0,0,0);
endfig;
```

To make more flexible use possible, a new type of expression is introduced. A `cmykcolor` is a quartet of numerics that behaves just like the already existing type `color`.

```
beginfig(1);
   cmykcolor cyan;
   cyan := (1,0,0,0);
   draw fullcircle withcmykcolor cyan;
endfig;
```

The new `cyanpart`, `magentapart`, `yellowpart` and `blackpart` allow access to various bits of a `cmykcolor` or the CMYK component of an image object.

Greyscale color model

There are only two new primitives for greyscale support: `withgreyscale` and `greypart`. That is because greyscale values are simple numerics.

```
beginfig(1);
   faded := 0.5;
   draw fullcircle withgreyscale faded;
endfig;
```

An image object cannot have more then one color model, the last `withcolor`, `withcmykcolor` or `withgreyscale` specification sets the color model for any particular object.

RGB color model

Two new aliases for the already existing RGB color model will be added to plain.mp. You are requested to use these new keywords `rgbcolor` and `withrgbcolor` when referring to the old color model.

Object specials

The new Metapost will support two specials that can be attached to drawing objects. They are output on their own lines, immediately before and after the object they are attached to.

The new drawing options are `withprescript` and `withpostscript`, their arguments simple strings that are output as-is. It is up to the macro writer to make sure that the generated Postscript code is correct.

```
beginfig(1);
  draw fullcircle
     withprescript "gsave"
       withpostscript "grestore";
endfig;
```

Standalone EPS

If `prologues` is set to the value 2, Metapost will generate a proper Encapsulated Postscript level 2 image that does not depend on dvips postprocessing. In this output mode, fonts not be downloaded, but their definition will be handled correctly (see the next paragraph).

Thanks to a detailed set of comments by Michail Vidiassov, this output mode will adhere to Adobe's Document Structuring Conventions. A private Postscript dictionary will be created to reduce the output size for large images.

Font re-encoding

If `prologues` is set larger then 1, any used fonts are automatically re-encoded. Their encoding vectors will be included in the output if that needed.

This code is based on the font library used by dvips and pdfTEX. Following in the footsteps of pdfTEX, there are two new associated primitives: `fontmapfile` and `fontmapline`. The string-value argument has the same optional flag that is used by pdfTEX:

	replace the current font list completely
+	extend the font list, but ignore duplicates
=	extend the font list, replacing duplicates
−	remove all matching fonts from the font list

```
prologues := 2;
fontmapfile "+ec-public-lm.map";
beginfig(1);
  draw "Helló, világ" infont "ec-lmr10";
endfig;
```

Font inclusion

Font inclusion is triggered by `prologues` being equal to 3. Whether or not actual inclusion / subsetting takes place is controlled by the map files. These can be specified using the syntax explained in the previous paragraph.

GNU groff support

Version 1.0 of Metapost will have native support for GNU groff, thanks to a set of patches by Werner Lemberg and Michail Vidiassov.

Future plans

The next release after this one is likely to contain the following:

- A option to build metapost as embeddable library instead of an executable.
- 64-bit internal calculations instead of the current 32 bits.
- Alternative output formats for easier parsing by script backends
- The possibility to store drawing objects
- 12-part transform expressions to make it easier for macro packages to implement three-dimensional points.

Where to find Metapost

WWW Homepage and portal:
http://www.tug.org/metapost

User mailing list:
http://www.tug.org/mailman/listinfo/metapost

Development & sources:
https://foundry.supelec.fr/projects/metapost

Taco Hoekwater

Invitation to EuroTEX2007

EuroTEX2007, April 28th until May 2nd, 2007, will be organized jointly by CS TUG, the Czechoslovak TEXUsers Group and GUST, the Polish TEXUsers Group, at Bachotek, near Brodnica, in north- east of Poland. This is the place where the annual GUST BachoTEXconferences are organized yearly since 1992. EuroTEX2007 will also be the XV BachoTEX, hence you might find it to be referred to as the EuroBachoTEX2007 conference.

We are trying to make it easier for you to get there: if justified by the number of pasengers, two EuroTEXbusses might be organized: one going from the south, from the Czech Republic or perhaps even from Hungary, and one from the west, from Holland. For details watch the conference site. The preliminary motto of the conference is

TEX: Paths to the Future

It seems to be justified by the recent developments which address the current shortcomings of our beloved system. These develop- ments will be presented during the conference and should make it very interesting. In no particular order, they are:

- new pdf TEXrelease,
- METAPOST v. 1.1,
- a batch of new font familes from the new project called TEXGyre,
- a working version of Omega 2,
- LuaTEX,
- XeTEX, and
- many more.

Watch the conference site:

`http://www.gust.org.pl/BachoTeX/EuroBachoTeX2007`

where you also can find a Google Maps pointer to the conference location. The site is going to be updated as new information be- comes available.

Please consider to contribute papers to make EuroBachoTEX2007 even more interesting. Dates for abstracts and paper submissions will be published soon at the conference web page.

And, of course: put this event into your calendar and then come and join the TEXies from around Europe and the world. You can not afford to miss it!

Jerzy Ludwichowski
(for the Organizing Committee)

Appendix G illuminated

Introduction

This paper aims to provide a collection of illustrations to *Appendix G* of *The TEXbook* [1].

To begin with, I will summarize briefly the main issues of *The TEXbook* which will be dealt with here; next, I confine myself to the explanation of the figures. Naturally, I will use the same notation as is used in *Appendix G*.

I recommend reading this paper simultaneously with *Appendix G*, although they partly overlap.

Motivation

TEX's algorithm for typesetting mathematical formulas is precisely described by Donald E. Knuth in *Appendix G* of *The TEXbook*. The description suffices to implement the algorithm in other languages. For example, it was implemented in JavaScript by Davide P. Cervone [2].

The only drawback of *Appendix G* is that no illustrations are provided. Of course, it is only a relative drawback. Professor Knuth apparently can live without illustrations. My comprehension critically depends on pictures. When they are missing in the original text, I end up making sketches while reading.

A few years ago, during my umpteenth reading of *Appendix G*, I prepared a bunch of sketches for myself. I didn't think about publishing them, as I was convinced that it is just my predilection or idiosyncrasy; but, judging from the paucity of available math fonts, I concluded that perhaps others might have similar problems.

Therefore, I decided to publish my illustrations to *Appendix G* in hope that they may prove useful, for example, for those working on math extensions for the available non-math fonts. Moreover, they may turn out to be helpful in future works on the improvement of TEX; after all, the algorithm is older than a quarter of century, and the world is not sleeping. For example, Murray Sargent III from Microsoft published recently (April, 2006) an interesting note on using Unicode for coding math [3]. He apparently was inspired by TEX: the notation is certainly TEX-based and well-known names appear in the acknowledgements and bibliography (Barbara Beeton, Donald E. Knuth, Leslie Lamport).

Table 1. Rules of the style change

Index styles

Basic	Superscript	Subscript
D, T	S	S'
D', T'	S'	S'
S, SS	SS	SS'
S', SS'	SS'	SS'

Fraction styles

Basic style of the formula α `\over` β	Numerator (α)	Denominator (β)
D	T	T'
D'	T'	T'
T	S	S'
T'	S'	S'
S, SS	SS	SS'
S', SS'	SS'	SS'

Math styles

In math formulas, the following eight styles are used:

- D, D' – in display formulas, generated out of text placed between double dollars `$$...$$` (*display style*);
- T, T' – in formulas occurring in a paragraph, i.e., placed between single dollars `$...$` (*text style*);
- S, S' – in formulas occurring in lower or upper indices, i.e., after the symbols `^` and `_` (*script style*);
- SS, SS' – in formulas occurring in indices of indices or deeper (*scriptscript style*).

Typically, in the plain format, 10-point fonts are used for the styles D and T, 7-point fonts for the style S, and 5-point fonts for the style SS. The "primed" styles, called cramped, use the same point sizes; they differ from the uncramped ones in the placement of subformulas. Table 1 defines the relations between styles of formulas and their subformulas.

Table 2. The types of atoms which may appear in a math list

Name	Description of the atom
Ord	ordinary, e.g., x
Op	holding a big operator, e.g., $\sum$
Bin	holding a binary operator, e.g., $+$
Rel	holding a relational symbol, e.g., $=$
Open	holding an opening symbol, e.g., $($
Close	holding an closing symbol, e.g., $)$
Punct	holding a punctuation symbol, e.g., ,
Inner	"inner", e.g., $\frac{1}{2}$
Over	holding an overlined symbol, e.g., $\overline{x}$
Under	holding an underlined symbol, e.g., $\underline{x}$
Acc	holding an accented symbol, e.g., $\hat{x}$
Rad	holding a radical symbol, e.g., $\sqrt{2}$
Vcent	holding a *vbox* produced by `\vcenter`

The symbol C will denote the current style, the symbol $C{\uparrow}$ the corresponding style of a superscript, and the symbol $C{\downarrow}$ the corresponding style of a subscript. From the rules given in Table 1 it follows that $C{\downarrow} = (C{\uparrow})'$.

Math lists

When TEX reads math material, it makes a *math list* out of it. The math list contains the following math-specific objects:

- *atom*, the basic element;
- *generalized fraction*, the result of `\above`, `\over`, etc.;
- *style change*, the result of `\displaystyle`, `\textstyle`, etc.;
- *boundary element*, result of `\left` or `\right`;
- *4-way choice*, the result of `\mathchoice`.

There are several types of atoms, depending on their contents. Table 2 (a duplicate of the table given on page 158 of *The TEXbook*) lists all possible cases.

Moreover, the math list may contain elements specific to a vertical list:

- *horizontal material* (*rules*, *penalties*, *discretionaries* or *whatsits*);
- *vertical material*, inserted by `\mark`, `\insert` or `\vadjust`;
- *horizontal space*, inserted by `\hskip`, `\kern`, or (acceptable only in math mode) `\mskip`, `\nonscript` or `\mkern`.

The math list is processed twice; after the second pass the "normal" vertical list is created. The scheme of the algorithm (consisting of 22 steps) is shown in Figure 1.

As we will see, font dimension parameters play an essential role. Following *The TEXbook*, I will denote the i^{th} parameter of the second family (that is, `\textfont2`, `\scriptfont2`, `\scriptscriptfont2`) by σ_i, and the i^{th} parameter of the third family (`\textfont3`, `\scriptfont3`, `\scriptscriptfont3`) by ξ_i. Note that the σ_i parameters have different values for different styles, while the values of the ξ_i parameters do not depend on the current style – when Computer Modern fonts and the plain format are used.

Selected steps of the algorithm

In most cases, the steps of the algorithm are straightforward and a short verbal explanation suffices. More detailed elaboration is needed, in my estimation, only for the typesetting of: radicals (step 11), mathematical accents (step 12), operators with limits (step 13), generalized fractions (step 15), and formulas with indices (step 18). The ensuing subsections deal with these steps.

Typesetting radicals

The process of assembling a formula containing a radical is presented in Figure 2. The top part of the picture shows the components (the radicand is typeset using the C' style), the bottom part the result of the assembling. Let w_x, h_x, d_x denote respectively the width, the height and the depth of the radicand, and w_y, h_y, d_y those of the radical symbol. The height of the radical symbol is expected to be equal to the thickness of the math rule, i.e., $h_y = \theta = \xi_8$. (A font designer may decide that $h_y \neq \xi_8$ but I cannot see the rationale for such a decision.) The quantity ψ is defined as follows:

$$\psi = \begin{cases} \xi_8 + \frac{1}{4}\sigma_5, & \text{styles } D, D', \\ \frac{5}{4}\xi_8, & \text{other styles.} \end{cases}$$

The quantity Δ is computed in such a way that the the radical symbol is vertically centered with respect to the radicand: $\Delta = \frac{1}{2}\big(d_y - (h_x + d_x + \psi)\big)$.

The baseline of the resulting formula coincides with the baseline of the radicand.

Typesetting mathematical accents

The typesetting of an accented formula is simpler than the typesetting of a radical. Nevertheless, Figures 3 and 4 reveal non-trivial subtleties of the routine.

As before, let w_x, h_x, d_x denote respectively the width, the height and the depth of the accentee (typeset in the style C'), and w_y, h_y, d_y those of the ac-

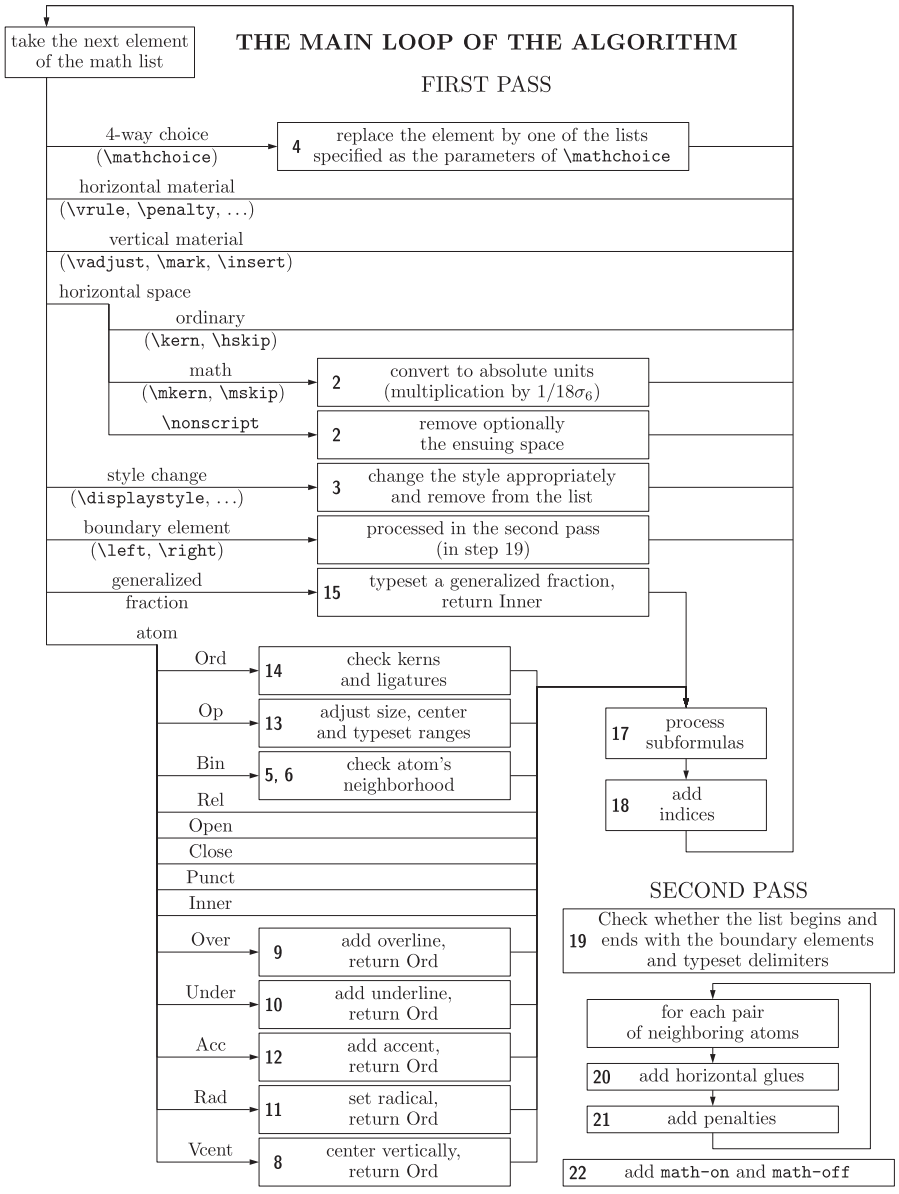


Figure 1. The scheme of the algorithm of the math list processing; the numbers at the left of the boxes refer to the steps of the algorithm, as described in *Appendix G*

center. Actually, these are the widths of the respective boxes; if the accentee is a symbol, its width, w_x, is computed as the sum $wd + ic$, where *wd* is the nominal (metric) width of the accentee, and *ic* the italic correction.

Both the accenter and accentee boxes are put into a *vbox* one above the other, and a negative vertical kern, $-\delta$, is inserted between the boxes, where $\delta = \min(\textit{x-height}, h_x)$. The *x-height* is defined by the fifth dimen parameter (`\fontdimen5`) of the accenter font.

The horizontal shift of the accenter, *s*, is equal to the implicit kern between the accentee and the special character, *skewchar* (defined by the command `\skewchar`); in the plain format, it is the character of code 127 (*tie after*) for family 1, and the character of code 48 (*prime*) for family 2. The kern has nothing to do with the shape of the `\skewchar`, but is intended to provide an appropriate correction due to the skew-

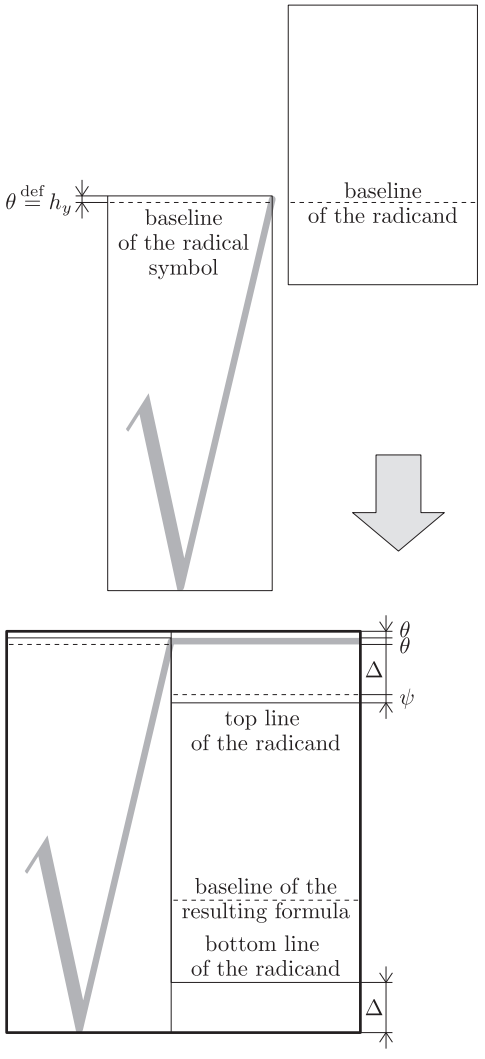


Figure 2. Assembling a radical; symbols explained in the text

ness of the accentee. If the accentee is already a boxed formula, TEX assumes that $s = 0$.

The width of the resulting formula is always equal to the width of the accentee, w_x; the baseline of the resulting formula coincides with the baseline of the accentee.

Typesetting operators with limits

The placement of the limits of an operator depends on the current style and the usage of `\limits` and `\nolimits` commands. If the style is D or D' and the operator `\nolimits` was not applied, the limits are placed above and below the operator, as displayed in Figure 5; otherwise, unless the operator `\limits` was

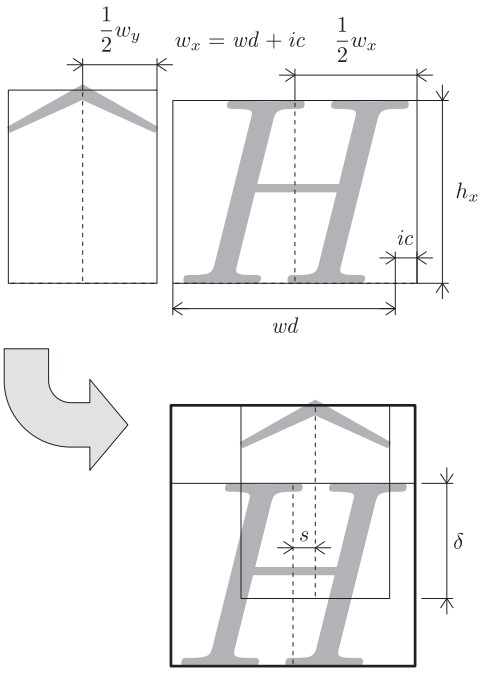


Figure 3. Assembling an accented formula, $w_y \leqslant w_x$;symbols are explained in the text

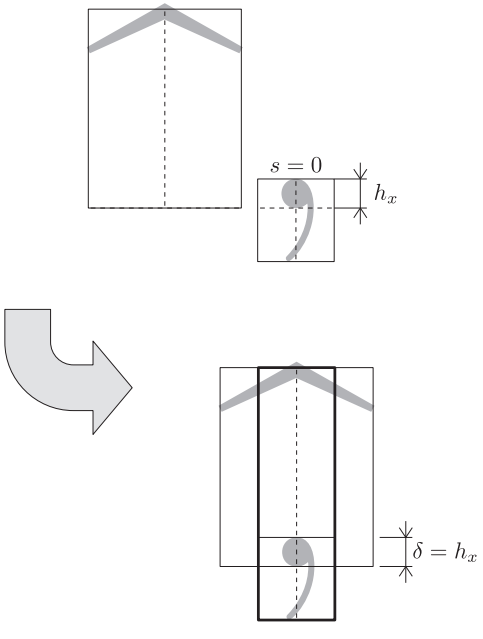


Figure 4. Assembling an accented formula, $w_y > w_x$;symbols have the same meaning as in Figure 3

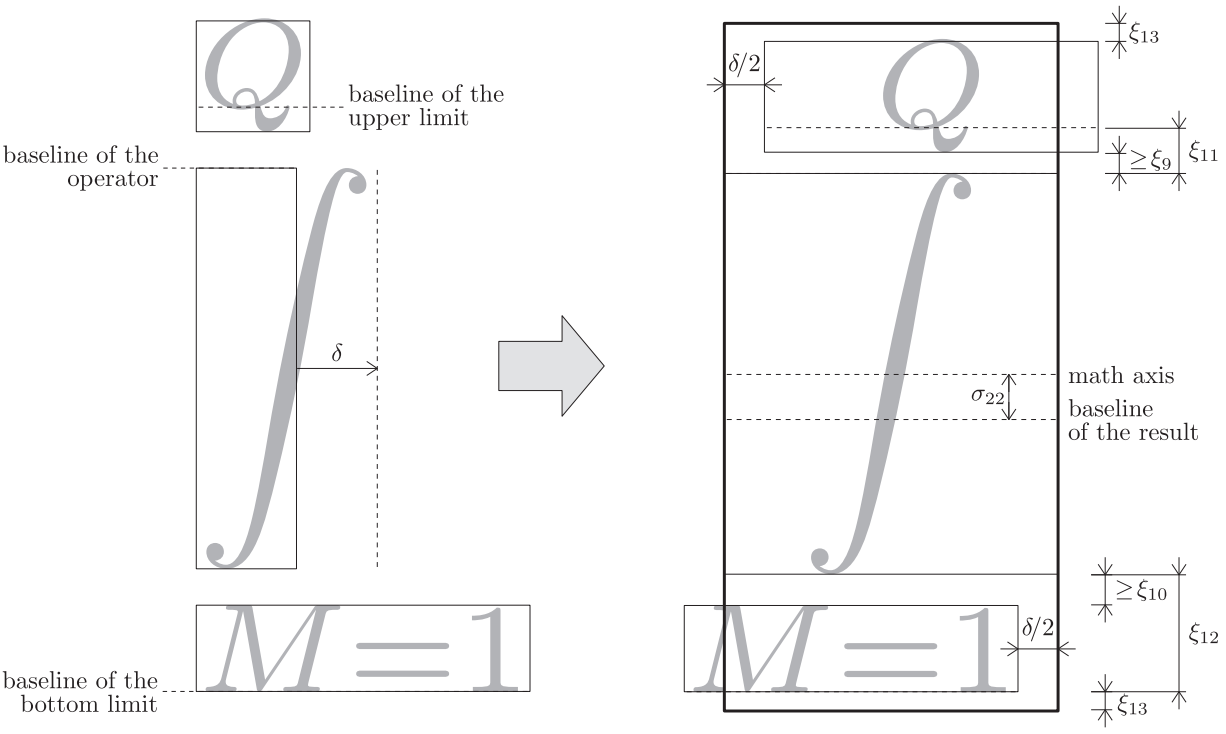


Figure 5. Assembling an operator with limits placed above and below; δ denotes the italic correction of the operator symbol

used, the limits are processed as fractions (see following section about fractions).

The operator symbol is centered vertically with respect to the math axis (σ_{22}). TEX tries to place the upper formula in such a way that its baseline is distant by ξ_{11} from the top of the operator; however, if the distance between the bottom of the upper subformula and the top of the operator would be less than ξ_9, the distance ξ_9 is forced. Similarly, the baseline of the lower subformula is distant by ξ_{12} from the bottom of the operator, unless the distance between the top of the lower subformula and the bottom of the operator would be less than ξ_{10}, in which case the distance ξ_{10} is forced.

For the correction of the horizontal placement of the limits, the value of the italic correction of the operator symbol (denoted by δ in Figure 5), is used.

Typesetting generalized fractions

There are two kinds of fractions implemented in TEX: with or without a bar between the numerator and denominator. They are typeset using different rules, as shows Figure 6. These rules, however, do not suffice, as the numerator and denominator are likely to collide. TEX cleverly avoids collisions, as is shown in Figures 7 and 8.

For a fraction with a bar, the numerator and denominator are shifted independently in order to provide a minimal gap, φ, between the formulas and the bar. The position of the bar remains intact – it coincides with the math axis (see Figure 7). For a fraction without a bar, a different strategy is used to avoid the collision, namely, both the numerator and denominator are shifted apart so that the gap between them is equal to φ (see Figure 8). Note that φ has a different meaning in the two cases.

Typesetting formulas with indices

The placement of indices is a fairly complex task due to the variety of situations that may occur.

Figures 9a – 9c show the horizontal placement of indices. If the kernel is a single symbol, the superscript, if present, is shifted to the right by the amount of the italic correction of the kernel symbol. Technically, a slightly different procedure is involved in the presence of a subscript, as stated in the description of step 17 in *Appendix G*. If the kernel is already a boxed formula, TEX assumes that $\delta = 0$. A kern of the value `\scriptspace` (s in the figure) is always appended to index formulas.

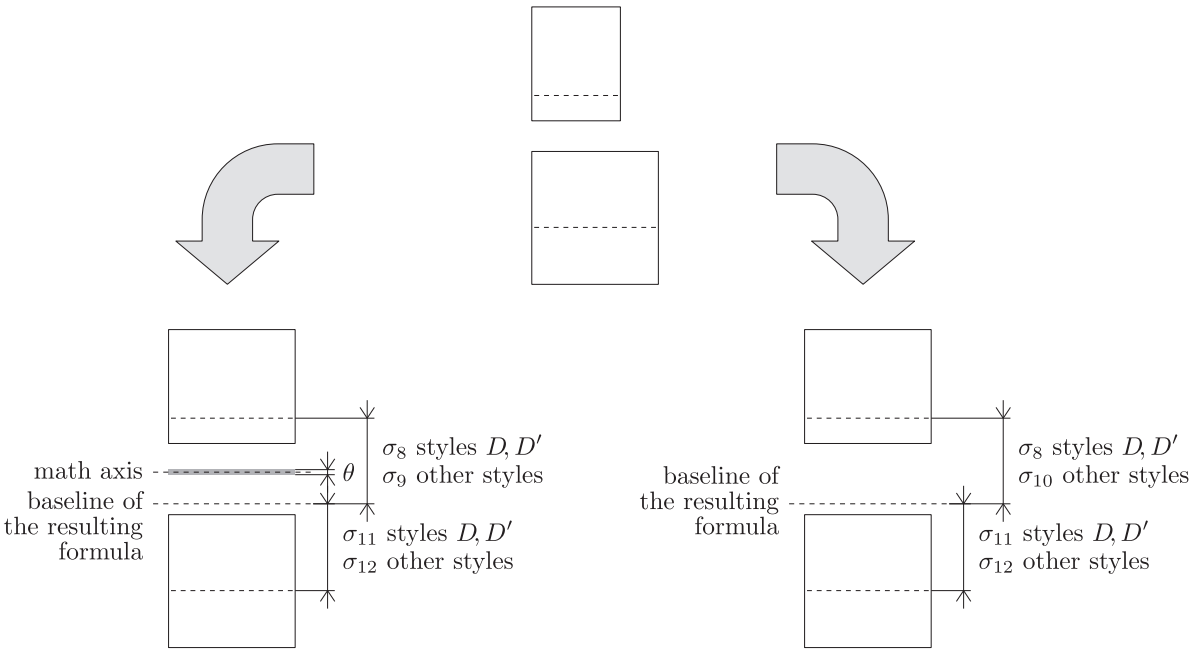


Figure 6. The placement of numerators and denominators in generalized fractions; the thickness of the rule, θ, is given either by the value of ξ_8 or explicitly; the latter possibility is provided by the `\abovewithdelims` command; observe that σ_9 is used for the formula with a bar, while σ_{10} for the formula without a bar

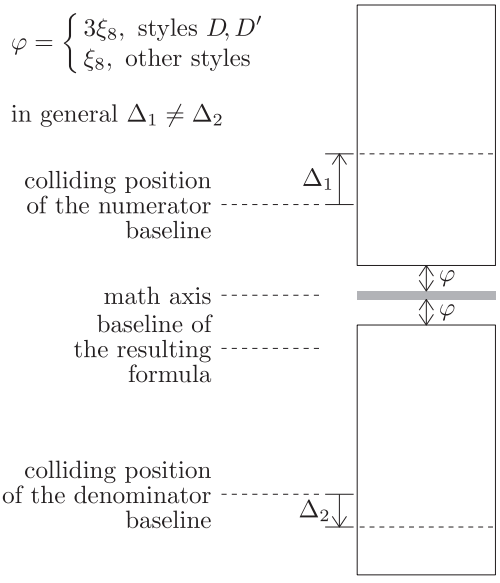


Figure 7:. Resolving a collision between the numerator and denominator in the case of a fraction with a bar

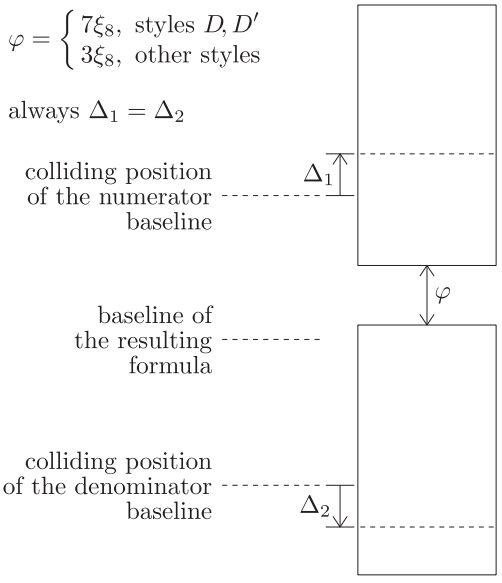


Figure 8:. Resolving a collision between the numerator and denominator in the case of a fraction without a bar

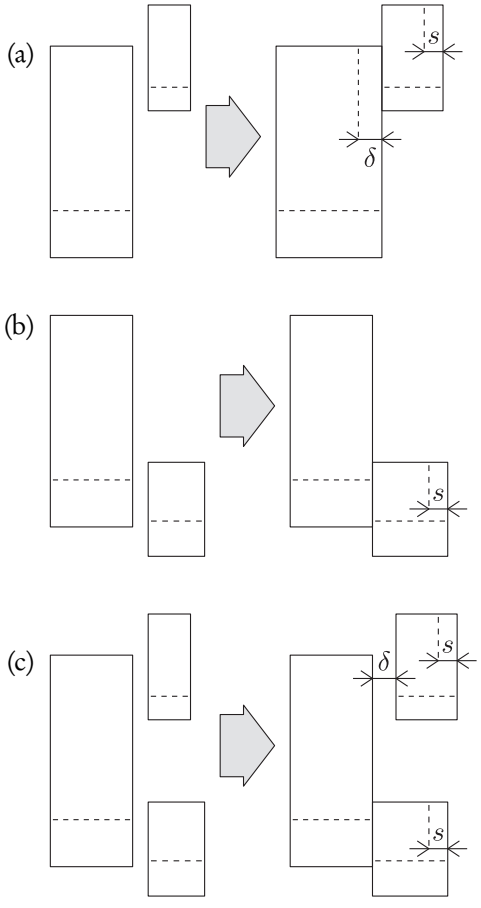


Figure 9. The horizontal placement of indices: (a) the placement of a lone superscript, (b) the placement of a lone subscript, (c) the placement of both superscript and subscript

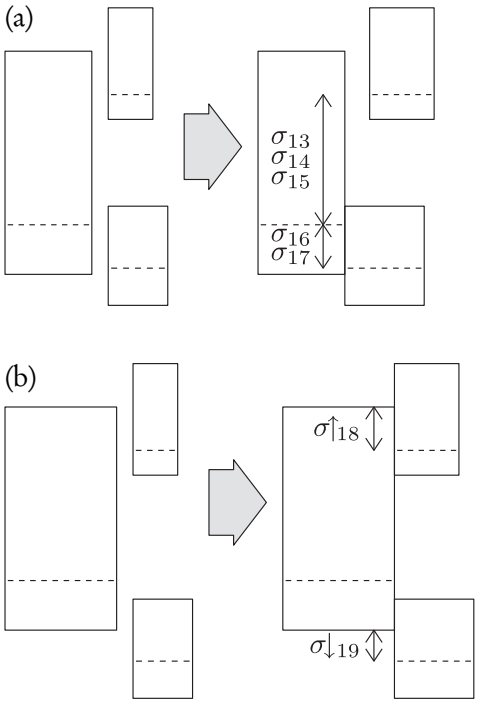


Figure 10. The vertical placement of indices: (a) the placement when the kernel is a symbol, (b) the placement when the kernel is a boxed formula

The procedure for the vertical placement (see Figures 10a – 10b) makes use of 7 parameters: from σ_{13} to σ_{19}. Again, different procedures are employed depending on the structure of the kernel. If it is a symbol, σ_{13} for the style D, σ_{14} for other uncramped styles, and σ_{15} for cramped styles are used for the placement of a superscript; for the placement of a subscript, σ_{16} is used if a superscript is absent and σ_{17} otherwise. If the kernel is a boxed formula, σ_{18} is used for the positioning of the superscript and σ_{19} for the positioning of the subscript. Moreover, the respective values are not taken from the current font: $\sigma\uparrow$ and $\sigma\downarrow$ mean that the parameters refer to the fonts corresponding to the styles $C\uparrow$ and $C\downarrow$, respectively.

Indices, as one can expect, are also subject to potential collisions. The actions for such a case are depicted in Figures 11a – 11d: (a) the bottom of the superscript formula cannot be placed lower than $\frac{1}{4}\sigma_5$ above the baseline; (b) the top of the subscript formula cannot be placed higher than $\frac{4}{5}\sigma_5$ above the baseline; (c) the gap between the bottom of the superscript and the top of the subscript cannot be smaller than $4\xi_8$ (recall that ξ_8 stores the math rule thickness) – the subscript is shifted if required; (d) finally, if the latter situation occurs, both indices can be shifted up so that the bottom of the superscript is not lower than $\frac{4}{5}\sigma_5$ above the baseline.

Conclusions

As originally mentioned, I prepared the illustrations initially for myself and only later decided to publish them in hope that somebody else may also benefit. Therefore, I eagerly welcome any feedback.

Acknowledgements

I am very grateful to Jerzy Ludwichowski and Piotr Strzelczyk for their prompt and willing help.

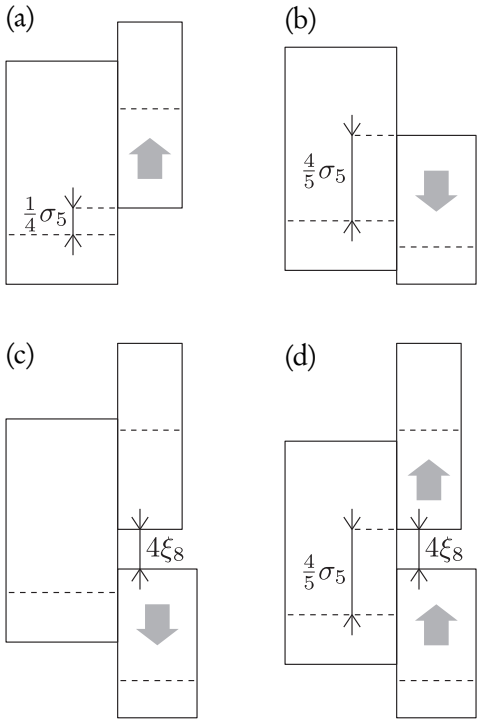


Figure 11. Resolving collisions of indices (further explanations in the text)

References

[1] Donald E. Knuth, *The TEXbook*, Computers & Typesetting: Volume A, Addison Wesley, 1986

[2] Davide P. Cervone, *jsMath: A Method of Including Mathematics in Web Pages*, `http://www.math.union.edu/~dpvc/jsMath/`

[3] Murray Sargent III, *Unicode Nearly Plain-Text Encoding of Mathematics*, `http://www.unicode.org/notes/tn28/`

Bogusław Jackowski

The New Font Project: TEX Gyre

Abstract
In this short presentation, we will introduce a new project: the "LM-ization" of the free fonts that come with TEX distributions. We will discuss the project objectives, timeline and cross-LUG funding aspects.

Introduction

The New Font Project is a brainchild of Hans Hagen, triggered mainly by the very good reception of the Latin Modern (LM) font project by the TEX community. After consulting other LUG leaders, Bogusław Jackowski and Janusz M. Nowacki, aka "GUST type.foundry", were asked to formulate the project.

The next section contains its outline, as prepared by Bogusław Jackowski and Janusz M. Nowacki. The remaining sections were written by us.

Project outline

Our aim is to prepare a family of fonts, equipped with a broad repertoire of Latin diacritical characters, based on the freely available good quality fonts. We think of an "LM-ization" of freely available fonts, i.e., providing about as many diacritical characters per font as we prepared for the Latin Modern font package (ca. 400 characters) which would cover all European languages as well as some non-European ones (Vietnamese, Navajo).

Since the provided character sets would be so close, such "LM-ized" fonts would work with all the TEX packages that the LM fonts work with, which would ease their integration. The result would be distributed, like the LM fonts, in the form of PostScript Type 1 fonts, OpenType fonts, MetaType1 sources and the supporting TEX machinery.

We would like to emphasize that the preparing of fonts in the OpenType format is an important aspect of the project. OpenType fonts are becoming more and more popular, they are Unicode-based, can be used on various platforms and claim to be a replacement for Type 1 and TrueType fonts. Moreover, Type 1 fonts were declared obsolete by Adobe a few years ago.

Since TFM format is restricted to 256 distinct character widths, it will still be necessary to prepare multiple metric and encoding files for each font. We look forward to an extended TFM format which will lift this restriction and, in conjunction with OpenType, simplify delivery and usage of fonts in TEX.

We especially look forward to assistance from pdfTEX users, because the pdfTEX team is working on the implementation on the support for OpenType fonts.

An important consideration from Hans Hagen: "In the end, even Ghostscript will benefit, so I can even imagine those fonts ending up in the Ghostscript distribution."

The idea of preparing such font families was suggested by the pdfTEX development team. Their proposal triggered a lively discussion by an informal group of representatives of several TEX user groups — notably Karl Berry (TUG), Hans Hagen (NTG), Jerzy Ludwichowski (GUST), Volker RW Schaa (DANTE) — who suggested that we should approach this project as a research, technical and implementation team, and promised their help in taking care of promotion, integration, supervising and financing.

The amount of time needed to carry out the task depends on the number of fonts to be included in the collection, but it can be safely estimated that a 4-font family (regular, italic, bold, and bold italic) can be prepared within 1–2 months, depending on the state of original material), which, for the collection of fonts mentioned below, would mean that the project can be accomplished within about two years. Assuming that the launch of the project could be made in the middle of 2006, the conclusion of the first stage of the project might be expected by the end of 2008.

The following fonts are presently considered worthy of enhancement:

1. The collection of 33 basic PostScript fonts, donated by URW++ and distributed with Ghostscript, consisting of eight 4-font families:
   - □ URW Gothic (i.e., Avant Garde)
   - □ URW Bookman
   - □ Century Schoolbook
   - □ Nimbus Sans (i.e., Helvetica)
   - □ Nimbus Sans Condensed (i.e., Helvetica condensed)

- □ Nimbus Roman (i.e., Times)
- □ Nimbus Mono (i.e., Courier)
- □ URW Palladio (i.e., Palatino)

and one single-font "family":

- □ URW Chancery (i.e., Zapf Chancery)

(Symbol and Zapf Dingbats fonts are left out.)

2. Donated by Bitstream (4-font families):
   - □ Charter
   - □ Vera
3. Donated by Adobe (4-font family):
   - □ Utopia
4. Other families donated by URW++:
   - □ Letter Gothic
   - □ URW Garamond

Perhaps other interesting free fonts will emerge in the future.

As to the budget — for the fonts listed above — there are altogether thirteen 4-font families and the 1-font URW Chancery. We would be satisfied if we could get a support of 1,500 EUR per a 4-font family and 500 EUR per a 1-font family. In the case of the fonts listed above this adds up to 20,000 EUR. We propose that the funding is made step-wise, i.e., the payments are made after the release of an enhanced family of fonts.

Obviously, there is scope for a second stage: the fonts can be further developed, as one can think for example of adding cap-small-caps, old style digits, proportional digits, and more. Once the first stage of the project is finished, we could embark on further enhancements. This would be somewhat simpler, but still a laborious task; we estimate the effort to be about 60–70% of the first stage.

Funding

The project can be divided into three stages:

- □ stage 1: combining existing fonts into LM compatible layouts and identifying gaps
- □ stage 2: filling in the gaps
- □ stage 3: math companion fonts

For these the estimated fundings needed are:

- □ stage 1: around 20,000 EUR
- □ stage 2: around 15,000 EUR
- □ stage 3: unknown

The exact figure for the second stage depends on what is needed and missing. We imagine the project to be extended with additional stages in order to bring more scripts into Open Type and/or to clean up other fonts as well.

The current financial state of the project:

- □ TUG India: 2,000 USD for the first year of the project and — if possible — a contribution for the next year.
- □ CSTUG: 500 EUR per year for the project duration, and expenses directly connected with implementing good Czech and Slovak support.
- □ NTG: 20% of the project costs, i.e., 4,000 EUR for the first stage.
- □ DANTE: 7,500 EUR this year; additional funding will be proposed at the September user group meeting.
- □ GUST: local infrastructure and expenses of font developers.
- □ TUG: positive but the amount is yet unknown, depends on contributions to the project fund, more info later this year.

Since the first stage runs this year and next year (start: May 2006) and since payments will take place after delivery of each completed font family, we can safely conclude that project expenses in 2006 can be covered and given the above we can also be confident that the rest of the first stage is secured.

The second stage is partly secured as well. We will try to broaden funding as soon and as much as possible. It would be good if this project can also take care of the needs of Indian, Greek, Cyrillic, Hebrew and Arab scripts, but we'll have to see …

The suite of fonts will be officially presented on a special font CD for members of user groups.

Co-ordination

This will done by DANTE e.V. in close co-operation with (at least) NTG, GUST, and TUG.

Late-breaking news: the name of the game

Nowhere above is explained the mysterious "TEX Gyre" which appears in the title. After this paper, consisting of the preceding sections, was presented at the BachoTEX 2006 conference, a discussion on how to name the collection of the New Font Project fonts was started by Bogusław Jackowski. A collective name for the project iself as well as names for the individual font

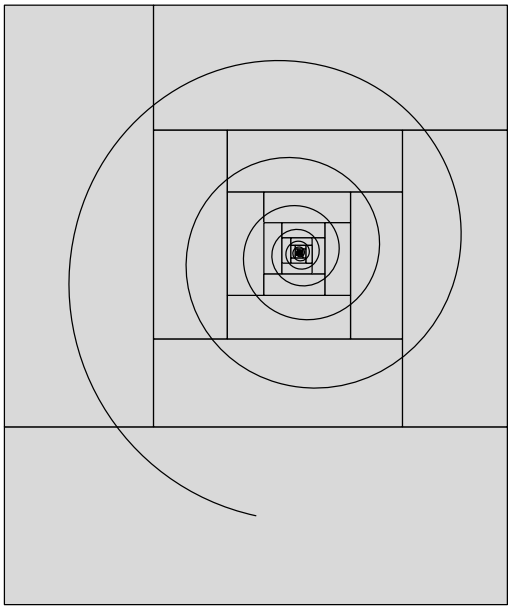


Figure 1. The TEX Gyre logotype

families were looked for. In another lively mail exchange among the above people and others, between 23rd May 2006 and 5th June 2006, the final proposal was hammered out.

Some of the collective names proposed: TEXModern, TEXSurge, TEXFountain, TEXSuper, TEX Font Foundry, TEXelent, not to mention stranger concoctions for individual families like TEXOnitalap (Palatino reversed) or — for Bookman reversed — TEXMankoob. A minimalist proposal of TEXF prevailed for a while, then TEX Fountain Collection fought against TEX Fount and even TEXFun was proposed until TEX Gyre was coined by Karl Berry.

This was liked by Bogusław: it nicely plays with the logotype he used in his presentations of the LM project, Figure 1. It is meant to symbolize the neverending striving for perfection as well as the "LM-ization" of the families.

The reader might not have previously encountered the word "gyre". For pleasure, here are a few dictionary explanations and examples of use in literature and on the web.

- the Collins Dictionary of the English Language defines "gyre" as:
  - a circular or spiral movement or path
  - a ring, circle, or spiral
- the Webster's Third New International Dictionary, Unabridged, says the following:
  - to cause to turn around: REVOLVE, SPIN, WHIRL; to move in a circle or spiral
  - circular motion by a moving body: REVOLUTION; a circular or spiral form: RING, VORTEX
  - in Scottish: a malignant spirit or spook
- more entertaining nearby terms from Webster's:
  - *gyre carline*, in Scottish: WITCH, HAG
  - *gyrencephalate*: a group of higher mammals: having the surface of the brain convoluted
- Wikipedia (`http://en.wikipedia.org/wiki/Gyre`) reports this:
  - a gyre is any manner of swirling vortex
  - W. B. Yeats uses the word in many of his poems, including "The Second Coming"

Table 1. Naming of the TEX Gyre fonts

Original URW name	PostScript name OpenType name TFM name root[a]
URW Gothic L	TeXGyreAdventor TeX Gyre Adventor qag
URW Bookman L	TeXGyreBonum TeX Gyre Bonum qbk
Nimbus Mono L	TeXGyreCursor TeX Gyre Cursor qcr
Nimbus Sans L	TeXGyreHeros TeX Gyre Heros qhv
URW Palladio L	TeXGyrePagella TeX Gyre Pagella qpl
Nimbus Roman No9 L	TeXGyreTermes TeX Gyre Termes qtm
Century Schoolbook L	TeXGyreSchola TeX Gyre Schola qcs
URW Chancery L	TeXGyreChorus TeX Gyre Chorus qzc

[a] For example, TFM files for the members of the Pagella family are named `qplr.tfm`, `qpli.tfm`, `qplb.tfm`, `qplbi.tfm` for the regular, italic, bold and bold italic faces, respectively. Encodings will be specified as a prefix such as `ec-`, as in Latin Modern.

– Lewis Carroll used the word as a verb in the opening stanza of his poem "Jabberwocky":

'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.

defining it as "to go round and round like a gyroscope"

- "The Widening Gyre", home of slistuk, the UK survival listserver (`http://dnausers.d-n-a.net/dnetIULU`), describes itself as "A site for the preparedness minded"
- Gyre, The Old Sow Whirlpool (`http://www.oldsowwhirlpool.com`), the biggest whirlpool in the Western Hemisphere, of over 70 meters in diameter, on the border of Canada and the United States, on the east coast of North America.

We will spare the reader's patience by not presenting the details of the discussion leading to the names of the individual font families. The general motivation when trying to find the names was that people dealing with fonts are likely to be familiar with the original PostScript names, so names resembling the original ones were desired.

We first tried to find the shortest English nouns that have some relationship to the original PostScript names, with possibly positive or neutral connotations, but this proved futile. In the end, Latin words were adopted. The result is presented in table 1 (check the meanings at, e.g., `http://archives.nd.edu/latgramm.htm`).

The families listed in items 2, 3 and 4 of the section "Project outline" have not yet been given TEX Gyre names. We are going to continue in the same spirit.

At the time of this writing (September, 2006), the Pagella and Termes families are at version 0.99 and should very soon be fully released, i.e., arrive at version 1.00. They can be found at `http://www.gust.org.pl/e-foundry/tex-gyre`. There is more to the fonts than was promised in the section "Project outline":

- over 1100 glyphs are available, including Cyrillic (though this was just carried over from the original without more ado);
- the complete Greek alphabet (for technical purposes rather than typesetting in Greek) is included — comments are welcome;
- cap-small-caps and old style digits are provided; this was initially planned for the second stage of the project.

Watch that space and enjoy!

P.S. XETEX (`http://scripts.sil.org/xetex`), an addition to the TEX family developed relatively recently by Jonathan Kew, shows that the decision to develop OpenType font versions was a good one: XETEX already uses the OTF version of the LM fonts. We hope that the TEX Gyre fonts will be no exception …

Hans Hagen
Pragma ADE
`pragma (at) wxs dot nl`

Jerzy B. Ludwichowski
Nicholas Copernicus University, Toruń, Poland
`Jerzy.Ludwichowski (at) uni dot torun dot pl`

Volker RW Schaa
GSI, Darmstadt, Germany
`v.r.w.schaa (at) gsi dot de`

The making of a (TEX) font

Abstract
We want to introduce a new display font to the TEX community. The font is a digitization of a series of Duane Bibby drawings, commissioned by Pragma ADE. The digital version for use with ConTEXt is prepared by Bittext based on scans provided by Pragma ADE.

Keywords
koeieletters, cow font, FontForge, Duane Bibby

Introduction

At TUG 2003 in Hawaii, Hans Hagen met with Duane Bibby. Hans was looking for some small images to enliven the ConTEXt manuals. A cutout of a very early sketch can be seen in figure 1, but it was soon agreed that consecutive drawings were going to be an alphabet.

Figure 1. The first drawing

Nothing much happened after that initial meeting, until the beginning of this year when Hans picked up the thread and got Duane started drawing. The alphabet quickly progressed. Starting in a rather naturalistic style like Duane's 'normal' TEX drawings, but later progressing toward a much more cartoon-like style, as can be seen from the drawings in figure 2.

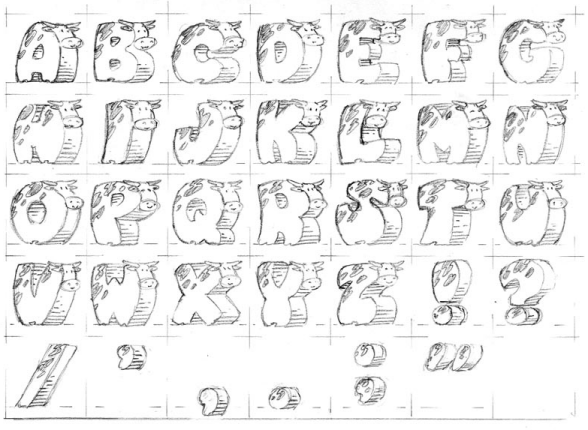


Figure 2. Rough design

For ease of use, it was clear that these drawings should ideally become a computer font. Taco Hoekwater agreed to take care of the digitization, and luckily the drawings were already prepared for that. As can be seen from the leftmost closeup in figure 3, the cows are drawn inside a grid. This ensures that they are all the same size, which is a vital requirement for a font.

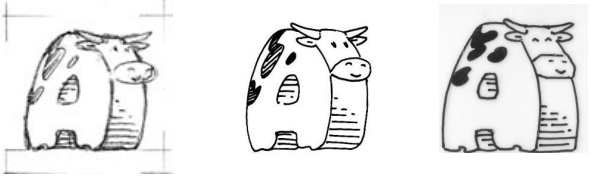


Figure 3. Closeups of the progressive design stages of the letter 'A'.

The center drawing in 3 is a still rather roughly inked version of one of the in-between drawings (there were many). In this particular one you can see that the mouth of the cow was originally more or less oval, but in the final form (on the right) it became much more hexagonal.

Digitization

The original sheets were sent to Pragma ADE by regular mail in the beginning of March. Hans scanned

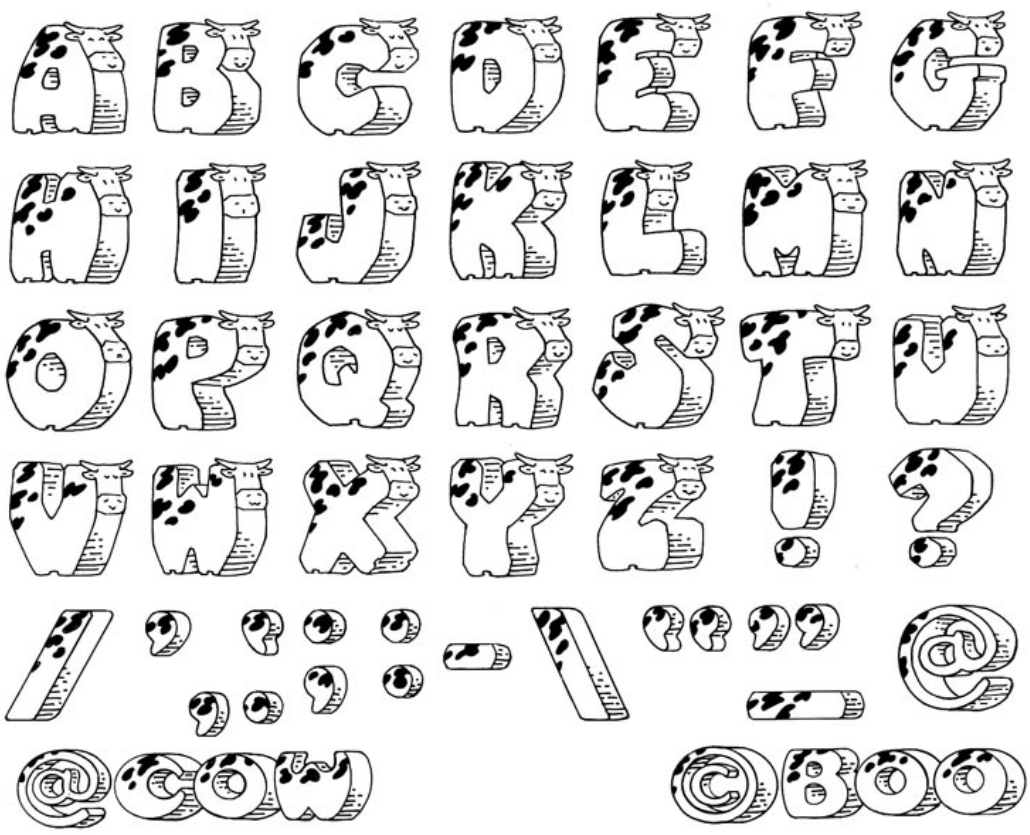


Figure 4. One of the original sheets, showing the alphabet and latin punctuation

the original sheets at 1200 dpi and then forwarded the images to Taco. There were four sheets in all, and one of them is shown in figures 4. The other three contain a number of TEX-related logos and a few (mathematical) symbols.

Preparing the images

The first task in the preparation of the font was to create a set of bitmap images for use by FontForge's import command.

For this, the four sheets had to be cut up into many smaller pieces, each containing a single glyph for the font. This being intended as a decorative font, the character set does not even contain the complete ASCII range. Nevertheless, almost a hundred separate images were created.[1]

FontForge automatically scales imported images so that they are precisely one em unit high. After cutting the sheets up to pieces, the images therefore had to be adjusted so that they all had the same height. Without that, it would have been nearly impossible to get all the drawn lines in the glyphs the same width. Figure 5 shows the adjusted version.

Figure 5. An imported bitmap image, with height adjusted

Automatic tracing

The autotracer in FontForge, which is actually the stand-alone autotrace program, does quite a good job of tracing the outlines. But, interestingly enough, only at a fairly low resolution. At higher resolutions it gets confused and inserts more than a quadratic amount of extra points for each resolution increase. Based on empirical tests, the images were scaled to 40% of their

original scanned size, resulting in bitmaps that were precisely 1000 pixels high.

As was to be expected, the autotracer brought out many of the impurities in the original inked version, as you can see in the left image of figure 6. Luckily, the amount of places where manual corrections like this were needed was not so big to force us to reconsider the digitization process.

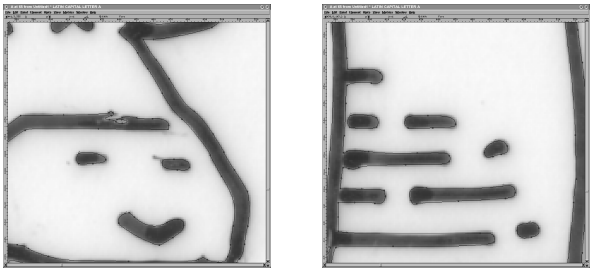


Figure 6. Close-ups of autotracer output

A more severe problem can be seen in the right-hand image of figure 6. The drawings contain hardly any straight lines. For a font of this complexity, it turned out to be absolutely necessary to simplify the curves. Without simplification, the rendering speed in PDF browsers became unbearably slow. All of the near-horizontal stripes in the bellies were manually removed and replaced by absolute straights.

Hinting

The final stage in the font editor is to add the PostScript hinting. A screen-shot of a manually hinted letter is visible in figure 7.

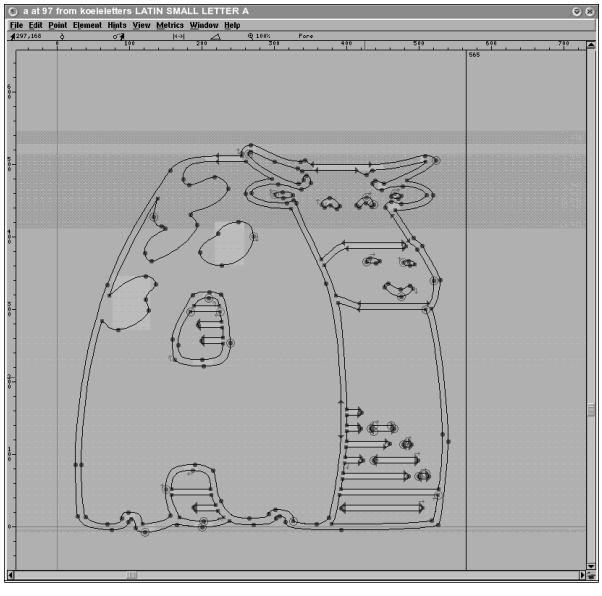


Figure 7. Finished outline

This part of the work in fact not completely finished yet. It is quite hard to find a balance between two possible extrema.

This part of the work is in fact turned out to be one of the largest jobs, because it is necessary to find a balance between two possible extrema.

On the one hand, if there are no hints at all, that results in nice small fonts that render quickly, but poorly.

On the other hand, if there is a lot of hinting information, that creates a much better appearance but it slows down the rendering. And sometimes extra hinting produces worse rendering than less hinting, especially with non-commercial renderers.

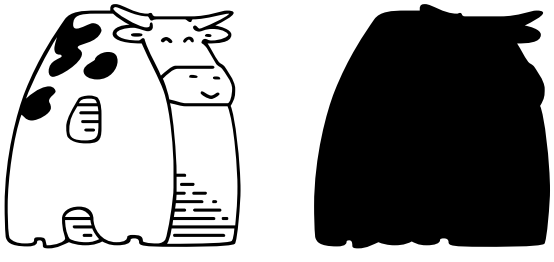
A middle ground can be reached, but unfortunately only by doing all hinting manually, and that took quite a lot of time.

Finishing the font

The font was saved as two separate PostScript Type 1 fonts, one with the text glyphs and one containing the logo glyphs. The text font is named 'koeieletters', the logo font 'koeielogos'. 'Koeieletters' literally translates from Dutch to English as 'cowcharacters', but the word 'koeieletter' is also is used to indicate a really big character. Like in a billboard, for instance.

Eventually it turned out that we needed a second set of two fonts. Sometimes you want to have text in the cowfont but on top of a colored background. The background would then shine right through the hide of the cow and that was of course unacceptable. Hence, we also have the fonts 'koeieletters-contour' and 'koeielogos-contour'.

Here is the final 'A', in the normal and the contour font:

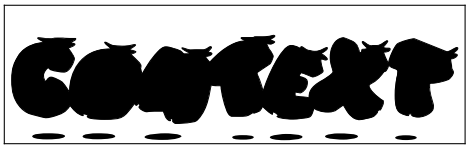


Playing around

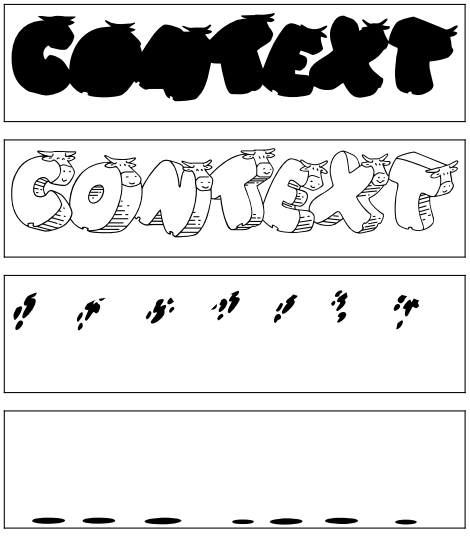
The original goal of this font was to enliven the Pragma ADE manuals. It would be a waste if we did not try to get the most out of the drawings provided by Duane, so we coded some rather silly effects in the font and its metrics. The final paragraphs of this article highlight a few of those.

Pragma ADE
The only lowercase symbols in the font are in the Pragma ADE logo itself:

The ConTEXt logo



If you look closely at the ConTEXt logo above, you can see that the shadows and the spots of the cows are drawn in different shades of grey (or colors). This is only possible because there are actually four characters involved instead of just the logo and the background contour:



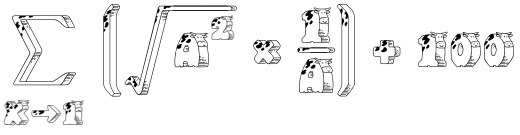
Logo ligatures
The line

can be input directly as:

```
Cows in ConTeXt
```

thanks to a handcrafted virtual font. This font contains a complete set of ligatures that travel from 'C at the beginning of a word', past 'e following the temporary ligature C_o_n_T', all the way to 't at the end of word'. Such word logos are defined for

Mathematics
As is true for the collection of normal text glyphs, the math set is also not very extensive. But there are just enough math symbols to allow some example math formulas to be created. Virtual fonts make sure that the input is what you expect from TEX.



Old-style Sheep
Seeing nothing but cows does tend to get boring after a while. To prevent the font from getting too predictable, we decided we needed some extra freshness. That is why the old-style numerals are actually sheep:

```
I count \oldstylenumerals{10} sheep
```

Final words

The koeieletter fonts can be downloaded from the Pragma ADE site: the font as well as the needed typescript file and macros are part of the standard ConTEXt distribution.

Taco Hoekwater Hans Hagen
Bittext, Dordrecht Pragma ADE, Hasselt
info (at) bittext (dot) nl

Je proefschrift in LaTeX zetten

Abstract
In dit artikel beschrijf ik hoe ik mijn proefschrift in LaTeX gezet heb. Ik ga in op mijn werkomgeving, de extra pakketten die ik gebruikte, de (lokale) truuks die ik in het bronbestand gebruikte en de problemen die ik ondervond (en oplossingen daarvoor).

Keywords
Proefschrift, memoir, apacite, LaTeX

Inleiding & werkomgeving

Een proefschrift schrijven is al behoorlijk uitdagende klus op zichzelf, al voordat je stil staat bij de manier waarop en het systeem waarin je je proefschrift wil gaan invoeren en laten zetten. De Sociale wetenschappen, waarin ik zelf recentelijk promoveerde, is een Microsoft gedomineerde markt waar het bepaald geen standaardkeuze is om je proefschrift in iets anders dan Microsoft Word te laten zetten. Al is het alleen maar om de samenwerking — via de inmiddels diep gewortelde 'Review'–functie van Word[1] — niet te hinderen met bestandsformaten waarvan de gemiddelde Sociale wetenschapper geen kaas heeft gegeten.

Ook na de beslissing om alle "goede" advies in de wind te slaan is het bepaald geen makkie om je proefschrift in LaTeX te gaan uitvoeren. Het internet wemelt van vele goed bedoelde maar bijzonder fragmentarische webpagina's vol adviezen die vaak op zeer lokale gebruiken en tradities georiënteerd zijn. Elke webpagina biedt wel een eigen style–file aan waarin alleen nog maar de inhoud gekopieerd hoeft te worden. Het converteren naar een bruikbaar formaat voor een Nederlandse universiteit is echter geen sinecure.

In dit artikel probeer ik enkele handgrepen te bieden voor de succesvolle promovendus die in het stadium van proefschrift schrijven is aangekomen en dit in LaTeX wil gaan doen. Dit is echter wel nadrukkelijk vanuit het perspectief van de Sociale wetenschappen gedaan. Het belangrijkste gevolg hiervan is dat andere normen gehanteerd worden bijvoorbeeld voor het zetten van de referenties in de tekst.

Daarnaast is het ook belangrijk te vermelden dat ik al jaren met LaTeX werk, maar dat ik eigenlijk nooit ingewikkelde klussen ter hand heb hoeven nemen. Dat wat onder LaTeX's motorkap zit is voor mij niet te vergelijken met de binnenkant van mijn broekzak. Een ervaren TeX–programmeur zal zich bij onderstaande ongetwijfeld wel eens op het hoofd willen krabben.

Voordat ik door het LaTeX–bronbestand van mijn proefschrift loop, geef ik eerst wat toelichting op de werkomgeving die ik hanteerde.

Werkomgeving. De keuze sec om een proefschrift in LaTeX te schrijven is er vaak eentje die al snel in de religieuze loopgraven uitgevochten wordt. Dit geldt net zo goed voor de gereedschappen waarmee je het werk uitvoert. Ik zal in het volgende vast wel een keer een gevoelige scheen raken; net zo vaak als, van de andere kant, een andere keuze achteraf meer gemak op had kunnen leveren.

Om te kunnen LaTeX'en heb je een distributie nodig waaruit je je classes en packages haalt. Voor mij is de TeXlive distributie al enige tijd het standaardsysteem op mijn computers. Dit hele artikel is dan ook gebaseerd op dit systeem; ik verwacht echter niet dat andere systemen zoals MikTeX volkomen anders werken. Om de meest belangrijke pakketten up to date te houden, installeerde ik de meest recente versie van de `memoir` class [1] en het `apacite` package [2] in mijn lokale TeX-boom. Als gebruiker van `beamer` voor mijn presentaties was ik al bekend met het PGF–package [4] waarmee je figuurtjes kan beschrijven in je LaTeX broncode. PGF is een variant op `ps_tricks` speciaal geschikt voor het produceren van PDF–figuren.

Een goede editor is ook van groot belang waarbij je vooral op je persoonlijke ervaring en voorkeur moet letten. Zelf gebruik ik al jaren Vim en dit was dus de vanzelfsprekende keuze om mijn proefschrift in te typen. Extra plugins, zoals bijvoorbeeld de Vim–LaTeX suite [3] en een (engelstalige) spellingchecker zullen je werk wat aangenamer maken (zie bijvoorbeeld `engspchk` [5][2]).

Tot slot installeerde ik ook nog Cygwin op mijn Windows XP machine. In eerste instantie omdat een Bash–shell toch wat efficiënter werkt dan de standaard XP command prompt; een automatische toetsencombi om mijn proefschrift in Vim te laten te compileren heb ik nooit echt goed voor elkaar gekregen. Bovendien bleek, in tweede instantie, dat het compileren van een proefschrift flink wat tussenstappen vereiste omdat ik natuurlijk een referentielijst moest opnemen (al dan niet geordend per hoofdstuk) en een auteursindex ook erg gewenst bleek. Het automatiseren van het compi-

leren leek mij veel gemakkelijker in een Linux–achtige omgeving dan onder Windows.

Om de compilatieproblemen op te lossen bestaan er vele verschillende oplossingen die met behulp van Google gemakkelijk gevonden kunnen worden. De oplossingen varieren van relatief simpele batchfiles tot zeer geavanceerde Ruby of Perl scripts en Makefiles. De Makefile die ik koos behoefde ook nog enige aanpassing, omdat ik via het apacite package een auteursindex wilde genereren die, zo bleek, in extra `.and` en `.adx` files resulteerde waarmee de oorspronkelijke Makefile geen raad wist. Uiteindelijk kon ik met één commando ofwel een DVI–file ofwel een PDF–file laten compileren. De gewijzigde Makefile is terug te vinden op mijn website [8]; dat geldt ook voor het basisbronbestand van mijn proefschrift en andere files die te lang zijn om in dit artikel integraal op te nemen.

Walkthrough

Een belangrijk aspect van mijn proefschrift is het feit dat ik de memoir class gebruikt heb als machine tussen mijn tekst en LaTeX. Eigenlijk wist ik niet van het bestaan van deze handige klasse af — Piet van Oostrum's hint over deze klasse in een eerder Maps artikel viel me pas achteraf op — totdat ik wat aan het rondsnuffelen was over het opmaken van boeken en daarbij het manual van deze klasse als een van de hits opdook. Oorspronkelijk had ik het proefschrift simpelweg willen maken met behulp van de standaard book klasse, maar het aanpassen van de (standaard) opmaak leek me een behoorlijke klus, zeker omdat ik die internals van TeX en LaTeX nog onder de knie moest krijgen. Memoir biedt echter de ideale oplossing als een extra niveau tussen TeX, LaTeX en de brontekst. Zo ongeveer elk oorspronkelijk commando van LaTeX wordt geherdefinieerd en voorzien van allerlei hooks die veel gemakkelijker zijn aan te passen in je bronbestand dan dat het is om de oorspronkelijke LaTeX–commando's aan te passen. Iemand die zich herkent in mijn plaats in de LaTeX–wereld kan ik deze klasse zeker aanraden!

```
\documentclass[a4paper,twoside,12pt,
openright,final]{memoir}
```

Het eerste commando[3] van een memoir document ziet er niet veel anders uit dan dat van een regulier `book`– of `article`–klasse document. De optie `final` heeft bij memoir, net als in alle standaard LaTeX klasses, de partneroptie `draft`. Wanneer je `draft` activeert, dan worden alle overfull hboxen gemarkeerd met een duidelijk zwart balkje in de marge en wordt het invoegen van figuren overgeslagen (althans, ze worden in de PDF/DVI alleen met een kader gemarkeerd). Hierdoor wordt het compileerproces soms merkbaar sneller afgerond.

Standaard packages. Na het inladen van de class de standaardpakketten. De echt standaard zaken heb ik niet genoemd (babel, graphicx, etc.); let echter op dat sommige pakketten in een bepaalde volgorde geladen moeten worden!

```
\usepackage{pgf}
\usepackage{booktabs}
\usepackage{europs}
\usepackage{wrapfig}
```

`pgf` laadt de nodige definities in voor de voorbereiding van het maken van PDF–tekeningen in je LaTeX–bronbestand. `booktabs` gebruikte ik om de tabellen in het proefschrift wat eleganter te maken. De voornaamste wijziging die dit pakket doorvoert betreft de spatiëring tussen lijnen en tekst. Daarnaast heeft het ingebouwde mogelijkheden om dikkere en dunnere lijnen te gebruiken.

Met `europs` wilde ik voorkomen dat mijn proefschrift vol zou komen te staan met "EUR" in plaats van €. Het `wrapfig` package heb ik op het allerlaatste moment opgenomen om één figuur die uiteindelijk kleiner uitviel dan de overige op een elegante positie in de lopende tekst in te laten vloeien. Er bestaan meerdere pakketten met deze doelstelling, maar wrapfig (b)leek op dat moment de simpelste keuze die aan mijn verwachtingen voldeed. Alle andere figuren heb ik steeds (uitsluitend) boven aan de pagina geplaatst (of op groot formaat op een aparte pagina) en de schaling zodanig aangepast dat er een prettige evenwicht ontstond tussen de tekst en de figuur met caption.

Packages met aanpassingen. Na deze standaardpakketten die weinig aanpassing nodig hadden heb ik de wat complexere pakketten met persoonlijke modificaties opgenomen.

Het is in mijn instituut gebruikelijk om een proefschrift te baseren op een serie van (gepubliceerde) artikelen. Om de relatie tussen artikel en hoofdstuk te definiëren wordt de hoofdstuktitel van een voetnoot voorzien die aangeeft uit welk(e) artikel(en) de tekst komt. Ik wilde hiervoor een aparte footnotemark gebruiken. Gewoon footnotes gebruiken en daarna doornummeren (eerste echte voetnoot wordt dan nr. 2) vond ik lelijk. Het nummer resetten was ook geen optie (twee noten met nr. 1?).

Ik koos voor een dagger, maar dan wel via standaard commando's omdat ik automatisch een uniforme voetnootopmaak wilde doorvoeren. Voetnoten zijn echter fragile waardoor in de sectioning commando's een `\protect` toegevoegd moet worden. Een extra complicatie was dat ik een zeer strakke cq. minimalistische (recto) openingspagina voor mijn hoofdstukken wilde creëren. De voetnoten moesten op de volgende verso–pagina belanden. Twee websites [6, 7] en het

footmisc package boden uitkomst.

```
\usepackage[stable]{footmisc}
\newcommand{\setsymbolfootnote}{%
  \def\thefootnote{\fnsymbol{footnote}}}
\long\def\symbolfootnotemark[#1]{\begingroup%
\protect\setsymbolfootnote\footnotemark[#1]%
  \endgroup}
\long\def\symbolfootnotetext[#1]#2{\begingroup%
\protect\setsymbolfootnote\footnotetext[#1]%
  {#2}\endgroup}
```

De nieuwe commando's komen van bovengenoemde websites waar ze eigenlijk gedefinieerd worden als (uitsluitend) een reguliere \footnote. Om de footnotemark en footnotetext te scheiden creërde ik dezelfde varianten, maar dan met een symbol als footnotemark.

Het laatste en belangrijkste package dat ik moest aanpassen was apacite van Erik Meijers. Citaties volgens de voorschriften van de American Psychological Association (APA) is de standaard in de Sociale wetenschappen. In eerste instantie gebruikte ik nog een behoorlijk oude versie, maar afgelopen jaar bracht Erik een tussenversie uit met behoorlijke uitbreidingen in functionaliteit.

```
\usepackage[emindex,bibnewpage,hyper]{apacite}
\renewcommand{\bibliographyprenote}%
  {\vspace{1em}}
\renewcommand{\bibliographytypesize}%
  {\footnotesize}
\newcommand{\Dutchvon}[2]{#2}
\renewcommand{\BOthers}[1]{et al.{}}
```

Volgens het manual moet dit pakket liefst als allerlaatste package geladen worden, maar in elk geval pas achter Babel. De opties geven aan dat ik een auteursindex wil die in de TOC opgenomen wordt; vergeleken met de andere index–opties zoals tocindex verzorgt emindex ook nog eens de titel van de auteursindex. Zet deze index op een nieuwe pagina met behulp van bibnewpage en 'misbruik' het bibliographyprenote om wat meer ruimte tussen header en tekst te krijgen. Gebruik de optie hyper voor compatibiliteit met het hyperref pakket dat ik wil gebruiken voor de online versie van mijn proefschrift.[4]

bibliographytypesize is een herdefinitie om de referentielijst in een kleiner font te zetten. Voor de auteursindex biedt apacite deze hook helaas niet waardoor ik de style file moest aanpassen door in de definitie van de emindex optie een \footnotesize in te voegen.

Het Dutchvon commando is een hack van Erik zelf waarmee je de referentielijst op een wat Nederlandsere manier kan sorteren; deze sortering is dan echter niet meer strikt volgens de APA norm. Dit commando vereist echter ook de nodige aanpassingen in de .bib–files en nog een verandering van Dutchvon net voor de referentielijst. Zie de documentatie van apacite voor meer info.

De laatste herdefiniering (\BOthers), tot slot van apacite, betreft een bug die ik ontdekte bij zeer fijnkorrelige inspecties van LaTeX's output. Volgens de normen van de American Psychological Association (APA) mag je bij een artikel met meer dan drie auteurs vanaf de tweede citatie in de tekst de 2e auteurs (en verder) vervangen door "et al.". Het pakket van Erik regelt dat helemaal automagisch, maar het bleek dat de punt achter 'et al' door LaTeX geïnterpreteerd werd als een sentence–ending dot met extra spatiëring. Dit probleem is met deze herdefinitie op te lossen, maar heeft als nadeel dat je je zinnen niet meer kan eindigen met een citatie, omdat hier dan ook de sentence–ending spacing zal verdwijnen. Ik heb me aan deze beperking kunnen houden, maar ik heb van Erik ook een stukje code waarin zelfs dit probleem met een extra commando opgelost wordt. Deze code heb ik op mijn website geplaatst [8]. Het resultaat van dit alles is opgenomen in de figuur met de openingspagina van mijn referentielijst.

Als laatste pakket heb ik hyperref ingevoegd om hyperlink navigatie toe te voegen voor de online versie van het proefschrift [8].

```
\usepackage[pdftex,pagebackref=true,
  linktocpage=true,breaklinks=true,
  bookmarks=true]{hyperref}
\hypersetup{%
  pdftitle={Congruency versus strategic
    effects in multimodal affective
    picture categorization},
  pdfauthor={Paul M. C. Lemmens},
  pdfsubject={Cognitive psychology and
    human-factors research},
  pdfcreator={PDFLaTeX (TeXLive 2003)
    and Vim},
  pdfproducer={PrintPartners Ipskamp,
    Enschede, The Netherlands},
  pdfkeywords={Stimulus-response
    compatibility, affective
    compatibility, affective-computing
    research},
  pdfpagemode={UseThumbs},
  pdfstartview=FitV,
  pdfpagelayout=SinglePage
}
\usepackage{memhfixc}
```

Ik heb hier niet lang over nagedacht en een setup overgenomen van een website (of misschien zelfs de officiële documentatie; ik weet het niet meer) die voor mij werkte en acceptabele resultaten gaf. Met de op-

ties bij het aanroepen van de style file geef ik aan dat LaTeX eventuele links die over een regelafbreekpunt heen gaan ook daadwerkelijk mag afbreken, dat er bookmarks aangemaakt moeten worden en dat er vanaf de TOC gelinkt wordt naar de openingspagina's van de hoofdstukken.

De meeste parameters van het `hypersetup` commando zullen ook wel voor zich spreken. De laatste drie geven aan hoe de PDF–lezer het PDF–bestand moet openen en weergeven. In dit geval met de thumbnails aan de linker kant, de pagina wordt verticaal passend op het scherm weergeven met één pagina per scherm. Jammer genoeg is de '2–page facing' optie van Acrobat geen optie, want daarvoor kon ik geen instellingen vinden.[5]

Het extra package `memhfixc` is een hack van de maker van memoir om dit laatste pakket compatibel te maken met hyperref. Memoir verandert zaken aan bijvoorbeeld de `\chapternumberline` commando's die hyperref niet kan hanteren. Dit extra package lost die problemen op.

Overigens heb ik de hyperlinks pas toegevoegd toen het officiële manuscript al lang bij de drukker lag. Dit simpelweg om 100% zeker te zijn dat de kaders die hyperref om de links tekent niet in de gedrukte versie van het proefschrift terecht zouden kunnen komen.

Persoonlijke macro's. Door memoir te gebruiken had ik uiteindelijk vrijwel geen complexe TeX–code meer nodig. Mijn macro's bestonden vooral uit een aantal simpele commando's die het typen van tekst wat vergemakkelijkten.

```
\newcommand{\schuinig}[1]{\textsl{#1}}
\newcommand{\Stim}{\schuinig{S}}
\newcommand{\Sr}{\Stim\ensuremath{_r}}
\newcommand{\Si}{\Stim\ensuremath{_i}}
\newcommand{\Resp}{\schuinig{R}}
\newcommand{\Response}{\Resp}
\newcommand{\R}{\Resp}
\newcommand{\SR}{\Stim--\Resp}
\newcommand{\SrR}{\Sr\,--\,\R}
\newcommand{\SiR}{\Si\,--\,\R}
\newcommand{\SiSr}{\Si\,--\,\Sr}
```

Ik nam een omgeving over uit de `apa.cls` om opsommingen in de lopende tekst mogelijk te maken.

```
\newcounter{lappieAPAenum}
\def\seriate{\begingroup%
   \setcounter{lappieAPAenum}{0}%
   \def\item{\addtocounter{lappieAPAenum}{1}%
     (\alph{lappieAPAenum})\space}%
   \ignorespaces}
\def\endseriate{\endgroup}
```

Mijn persoonlijke trots is nog altijd het `\marginnote` commando dat ik zelf in elkaar gestoken heb met veel proberen en optimaliseren. Omdat mijn werkmodus vaak onderbroken wordt door gedachtes die ik heb over stukken tekst die ik typ — wijzigingen, alternatieve formuleringen, aandachtspunten, et cetera — gebruikte ik vaak het `marginpar` commando om die gedachtes snel vast te leggen. Al snel werden dit opmerkingen die ik zelf al na twee dagen niet meer snapte omdat de context ontbrak.

```
\newcounter{marginnotecounter}%
\setcounter{marginnotecounter}{1}%
\newcommand{\marginnote}[1]{%
\def\baselinestretch{0.9}%
\textbf{(\themarginnotecounter)}%
\marginpar{%
\parbox[t]{20mm}%
{\raggedright\tiny\textsf{%
  (\themarginnotecounter) #1}}}%
\addtocounter{marginnotecounter}{1}}%
```

Dus definieerde ik een nieuwe teller die een positie in de tekst markeerde en vervolgens in de marge de noot en teller plaatste. Maar wanneer ik, wederom als verplichting van de APA, m'n regelafstand verdubbelde voor manuscripten, dan werd die regelafstand in de margenoten meegenomen. In principe van weinig belang, omdat die opmerkingen toch vanzelf zouden moeten verdwijnen voordat een manuscript definitief zou mogen worden, maar het leverde visueel erg horkerige DVI's op. Daarom voegde ik in de laatste versie een lokale `baselinestretch` aanpassing door en verkleinde ik het font. Een voorbeeld van een margenoot is te zien in figuur 1, aan het eind van dit verhaal.

Opmaak van het manuscript. Toen de functionaliteit van het bronbestand eindelijk geregeld was, kon ik aan de opmaak beginnen. Eigenlijk had ik dit alles natuurlijk in een stylefile moeten gooien, maar dat is een stap waar ik nooit meer aan toe gekomen ben. Veel van deze aanpassingen heb ik pas doorgevoerd toen het manuscript inhoudelijk helemaal af was en tot drie keer toe gecontroleerd was op typefouten en andere onhebbelijkheden. Pas dan heb je je manuscript namelijk helemaal af zodat je echt de fijne details van de opmaak kan gaan regelen. Maak niet de fout om deze fijnkorrelige details te vroeg te willen regelen, want dan kom je uiteindelijk in een vicieuze cirkel terecht — ik spreek uit ervaring!

```
\addtolength{\textheight}{17pt}
\checkandfixthelayout
```

De eerste aanpassing die ik doorvoerde was een pragmatische: ik kwam op 145 pagina's uit wat één pagina meer was dan dichtsbijzijnde 16–voud[6]. Het verhogen van de tekstheight met 17 punten was precies genoeg om één pagina van het totaal aantal af te snoe-

pen. Het commando \checkandfixthelayout is een min of meer verplicht memoir commando dat na zulke wijzigingen er voor zorgt dat allerlei gerelateerde parameters op de juiste manier aangepast worden. Als ik me goed herinner wordt er ongeveer een heel hoofdstuk in het memoir manual aan de paginaopmaak gewijd; je zult het mij niet kwalijk nemen dat ik dat hier niet ga herhalen!

```
\usepackage{setspace}
\newcommand{\BaseLineStretch}{1.10}
\setstretch{\BaseLineStretch}
```

Binnen de Sociale wetenschappen, maar vermoedelijk ook binnen andere richtingen is het gebruikelijk om manuscripten met een dubbele regelafstand in te dienen. Sommige promovendi hanteren deze norm zelfs bij het opmaken van de uiteindelijk (te drukken) versie van hun proefschrift, maar dat vond ik niet mooi. Van de andere kant vond ik de leesbaarheid met regelafstand van 1 te laag en na lang proberen en laten vergelijken door "naïve" lezers kwam ik op een afstand van 1.10 of 1.15 uit. Ik hield de 1.15 achter de hand voor het geval dat ik het pagina–aantal zou moeten verhogen naar een 16–voud.

Door trial en error ontdekte ik overigens een eigenaardige, maar achteraf gezien logische, interactie tussen het veranderen van de baselinestretch en de \\[-3mm] commando's die ik gebruikte in een PGF–plaatje (zie figuur 2.2 in mijn proefschrift.) Voor een artikelversie van een bepaald hoofdstuk, dat op dubbele regelafstand moest, had ik de negatieve afstand in het \\ commando nodig om de tekst in het PGF–plaatje (verticaal) juist uitgelijnd te krijgen. Toen ik de regelafstand weer verkleinde voor het proefschrift was er natuurlijk opeens teveel negatieve afstand. Het duurde even voordat ik deze interactie in de gaten had.

```
\usepackage{palatino}
\usepackage{pxfonts}
```

Iedereen die vaker LaTeX heeft gezien of gebruikt herkent het in één oogopslag: het is ofwel de paginaindeling ofwel het font dat een LaTeX–gebruiker verraadt. De pagina-indeling was al voldoende anders dus bleef alleen het font over om nog te wijzigen. Ik hanteerde twee mogelijke varianten: Bookman of Palatino. Veel van mijn proeflezers vonden de sample van pagina's in Bookman leuker en prettiger lezen. Bookman ziet er iets dikker uit wat wel een bepaalde aantrekking had. Zelf was ik echter nog net iets meer gecharmeerd van Palatino. Dat feit gecombineerd met een forse verhoging van het totaal aantal pagina's bij gebruik van Bookman deed de keuze uiteindelijk op Palatino vallen.

Gelukkig merkte ik nog net op tijd dat het standaard package palatino de fonts in mathmode niet mee deed veranderen. Via TEX–NL werden de suggesties mathpazo en pxfonts gegeven. Puur op visuele basis koos ik uiteindelijk voor pxfonts om het Palatinofont mooi in de mathmode door te laten lopen.[7] Ik heb geen gekke sprongen meer gedaan om nog de oldstyle cijfers te realiseren (al had ik dat misschien wel mooi gevonden), want daarvoor was de tijdsdruk te groot.

Het volgende aandachtspunt voor de opmaak was de openingspagina van de hoofdstukken. Zoals ik al eerder aangaf wilde ik deze zo clean mogelijk houden: een cijfer, de titel en eventueel nog een quote in een benedenhoekje.

```
\makeatletter
\makechapterstyle{companion}{%
  \renewcommand{\afterchaptertitle}{%
    \thispagestyle{empty}}
  \renewcommand{\chapnamefont}{%
    \normalfont\LARGE\scshape}
  \renewcommand{\printchaptername}{%
    \raggedleft\chapnamefont \@chapapp}
  \renewcommand{\chapnumfont}{\normalfont\Huge}
  \setlength{\chapindent}{\marginparsep}
  \addtolength{\chapindent}{\marginparwidth}
  \renewcommand{\printchaptertitle}[1]{%
    \begin{adjustwidth}{}{-\chapindent}
      \raggedleft \chaptitlefont ##1\par\nobreak
    \end{adjustwidth}}
}%
\makeatother
\pagestyle{companion}
\chapterstyle{companion}
```

Ik had al besloten dat ik de door memoir voorgedefinieerde companion–emulatie voor pagestyle en chapterstyle wilde gebruiken. Het paginanummer op de openingspagina van het hoofdstuk wilde ik echter verwijderen. Het lukte me niet om dit met een of andere hook van memoir te corrigeren, dus besloot ik simpelweg de gehele definitie uit de classfile over te nemen en naar believen aan te passen. Let op de \makeatletter en \makeatother die nodig zijn wanneer je een \@–commando gebruikt. Als ik een stylefile gebruikt had waarin deze set van commando's was opgenomen, dan had ik de \makeat* commando's achterwege kunnen laten. Daarna kon ik veilig de pagestyle en chapterstyle voor de rest van het proefschrift vastleggen. Figuur 2 laat een voorbeeld zien van een hoofdstuk-pagina.

Toen ik besloot in de TOC uitsluitend de hoofdstukken aan te geven (\maxtocdepth{chapter}), bleek dit erg lelijk te worden. Het vet van het nummer en de titel van de hoofdstukken werd niet onderbroken door lijstjes van niet–vet gedrukte sections in die hoofdstukken. Met behulp van \cftchapterpresnum en \cftchapterfont kon ik het nummer van het hoofd-

stuk wel in het vet krijgen, maar bleef de titel van het hoofdstuk netjes in normale tekst staan.

```
\renewcommand{\cftchapterpresnum}{\bfseries}
\renewcommand{\cftchapterfont}{}
\maxtocdepth{chapter}
```

Natuurlijk is een goed proefschrift doorspekt van min of meer diepzinnige quotes van min of meer beroemde mensen. Memoir biedt daarvoor het epigraph mechanisme. Voor mijn proefschrift paste ik de default breedte en positie aan.

```
\setlength{\epigraphwidth}{0.5\textwidth}
\epigraphposition{flushleft}
```

Voor het zetten van een aantal (sets van) plaatjes gebruikte ik in eerste instantie het `subfig` pakket. Toen ik echter met de opmaak van de captions ging spelen bleek dat subfig en memoir elkaar niet aardig vinden. Aangezien memoir en de wijzigingen aan de caption prioriteit hadden moest subfig wijken. Het manual van subfig (of een ander subfig–achtige) gaf een goede hint om een bepaalde opmaak met subplaatjes ook met gewone TEX–commando's voor elkaar te krijgen. Daarna was het nog slechts een kwestie van de juiste hooks aanpassen om het font te verkleinen en (dus ook) de afstand tussen de broodtekst en de captions te verkleinen.

```
\captionnamefont{\footnotesize}
\captiontitlefont{\footnotesize}
\setlength{\textfloatsep}{35pt}
```

Inhoud. Met deze aanpassingen kon ik het opmaken van het manuscript afsluiten. In het vervolg van dit artikel wil ik een aantal (memoir) commando's uitlichten die ik in de inhoudelijke tekst van het proefschrift kon gebruiken. Op de standaardzaken zoals `\title` `\author` etc. zal ik niet ingaan; net zomin op de `\frontmatter`–commando's. Het verdient wel de aanbeveling om de theoretische hoofdstukken van het memoir manual eens door te lezen over de indeling en de opzet van de zogenaamde half–title page, frontispiece/copyright page enz.

Toen ik, op een verder lege pagina, de copyright mededelingen volledig aan de onderkant van de pagina wilde plaatsen, ondervond ik dat `\vfill` niet zomaar de grootst mogelijke verticale afstand opeet. Voor mij was het een verrassing dat er 'tekst' voor dit commando moet staan. De oplossing dient zich dan relatief eenvoudig aan: '`\ \vfill`".

Ik merkte ook door trial and error dat wanneer ik de lange titel van het proefschrift in een groter font wilde zetten op de pagina, dat dan (klaarblijkelijk) de regelafstand niet aangepast wordt. Het eindresultaat is dan dus een aantal regels dat (verticaal) veel te kort op elkaar staat. Hier kwam het `setspace` package goed van pas. Een beetje tweaken leverde een regelafstand van 1.35 op voor een `\textbf{\Large{titeltekst}}`.

Bij het bekijken van de titelcode kwam ik nog een zogenaamde FIXME tegen. Het is een markering die ik gebruik om incomplete stukken tekst, broncode of anderszins te markeren voor latere analyse.

```
%%FIXME: uitlijning op 1 regel!
een wetenschappelijke proeve op het gebied %%
  van de Sociale Wetenschappen
```

In dit geval had ik al geconstateerd dat in de PDF het woord Wetenschappen een regel lager stond dan de rest van de zin. Klaarblijkelijk heb ik deze FIXME uiteindelijk gemist bij mijn controles (zie de figuur met het titelblad van mijn proefschrift[8]) en natuurlijk pas net na het accorderen van de drukproeven weer opgemerkt. Om dit soort situaties te voorkomen heb ik eens een pakket gevonden dat een commando implementeerde dat vergelijkbaar is met mijn `\marginnote`, met de aanvulling dat aanwezigheid van marginnotes het compileerproces onderbreekt. De naam en plaats van dit pakket heb ik helaas niet meer kunnen achterhalen.

Na deze (door de universiteit verplichte) pagina's dan toch eindelijk de wetenschappelijk inhoud. Elk hoofdstuk heb ik met een `\include` in het hoofdbestand opgenomen. Hieronder licht ik het voor dit publiek interessantste gedeelte uit.

```
\chapter[Stimulus--Response Compatibility and
  Affective Computing: A Review][Stimulus--%
  Response Compatibility and Affective
  Computing]{Stimulus--Response Compatibility
  and Affective Computing: A Review
  \symbolfootnotemark[2]}\label{ch:backgr}%
  \label{ch:background}
```

Een belangrijke reden om memoir te gebruiken, behalve de handige hooks, was het `\chapter`–commando dat maar liefst drie argumenten kent. De standaard LATEX–versie kent er maar twee: een verplicht argument en een optioneel argument voor in TOC. Het verplichte argument wordt echter ook gebruikt in de header van de pagina. Meestal dus veel te lang, waardoor de tekst gaat overlappen. Hierboven dus drie titels: de eerste voor in de TOC, de tweede voor in de header en de derde als echte titel van het hoofdstuk. In dit derde argument kom je ook weer het `\symbolfootnotemark`–commando tegen dat ik eerder definieerde om een lege, minimalistische hoofdstukopeningspagina te creëren.

```
%% Epigraph!?
\vfill
```

```
\epigraph{I often say ...}%
  {\textit{Lord Kelvin (1824--1907)}}

\cleartoverso
\section*{Abstract}

\input{intro-review-abstract}

\symbolfootnotetext[2]{This chapter is
  submitted as: ...}

\cleartorecto
\section{Introduction}

\input{background}
```

Met het `\cleartoverso` gaan we naar de 'achterkant' van deze openingspagina (althans in de uiteindelijke dubbelzijdige versie van het proefschrift) waar we het abstract afdrukken, de voetnoot plaatsen en daarna weer doorschuiven naar de volgende rechterpagina om het hoofdstuk (inhoudelijk) te beginnen. Het resultaat van deze LaTeX–code (incl. het `\chapter[][]{}`–commando in de vorige paragraaf) is te zien in de figuur met daarin de openingspagina van hoofdstuk 2 van mijn proefschrift.

In een bepaald hoofdstuk wilde ik naderhand nog wat controle–studies toevoegen die echt specifiek bij dit hoofdstuk hoorden. Ik verwachtte dat ik een rare constructie zou moeten gaan bedenken om de appendices niet meer numeriek te laten nummeren, maar ook hier kwam memoir weer als onverwacht behulpzaam om de hoek.

```
\begin{subappendices}
\end{subappendices}
```

De `subappendices` omgeving kun je gebruiken om (eenmalig) één of meerdere appendices achter een hoofdstuk in te voegen, in plaats van op het einde van een boek. Het zorgt voor een andere naamgeving ("Appendix") en een alfabetisch telling (bijv. "4B") van de secties. Met behulp van het `\section`–commando kun je verschillende appendices opnemen in de subappendices omgeving.

Backmatter

En dan na heel veel typewerk en het gebruikelijke schrappen kom je aan de laatste loodjes. De `backmatter` van je proefschrift. Omdat dit een apart gedeelte is van het document heb ik hier veiligheidshalve de `\pagestyle{companion}` en `\chapterstyle{companion}` herhaald.

Omdat ik wilde dat de referentielijst (via `\renewcommand{\refname}{References}`) wat kleiner gezet werd en wat gecomprimeerder op de pagina moest komen (denk weer aan die veelvoud van 16 pagina's) heb ik enkele van de nieuwe `apacite`–hooks gebruikt.

```
\setstretch{1.0}
\setlength{\bibindent}{-1em}
\setlength{\bibitemsep}{+\parsep}
```

Als toetje ook nog een auteursindex omdat ik geen referentielijsten per hoofdstuk heb geprint: `\printindex[autx]`. Om voor deze lijst dezelfde regelafstand en met name fontgrootte te kunnen handhaven moest ik helaas wél in de `.sty`–file van apacite hacken.

Lastige figuur

Ik moest een tamelijk complexe figuur maken om experimentele resultaten te verhelderen; het complexe betrof vooral het inlezen en plaatsen van mijn stimuli in die figuur. Mijn statistiek deed ik voornamelijk in R [9] dat alleen pixmap kent als een grafisch formaat dat binnen een gegenereerde figuur geplaatst kan worden.[8] Wegschrijven van een figuur in verschillende formaten is in R geen probleem: PNG, JPG, EPS en PDF behoren alle tot de mogelijkheden. De inhoud van de figuur bestond uit geïmporteerde plaatjes, tekst en gegenereerd grafisch werk (zie pagina 70 van mijn proefschrift [8]).

Nadeel van wegschrijven in PNG is dat de figuur niet echt lekker meer wil op–/neerschalen met name omdat de fonts dan lelijk worden (idem voor JPG). PDF was op dat moment geen optie omdat het me niet lukte om de fonts in de resulterende file op te nemen; dat betekende dat de PDF van mijn hele proefschrift niet door de preflight van Adobe zou komen. EPS bleek echter ook niet handig omdat al die kleine plaatjes die ik invoegde uiteindelijk tot een EPS bestand van ruim 50Mb samengevoegd werden. Dat werkte niet handig met het verder verwerken van de PDF van het hele proefschrift.

Via TEX–NL kreeg ik een groot aantal tips toegezonden die varieerden van compressietechnieken tot allerlei conversiepaden. Gelukkig bleek ik deze oplossingen niet meer nodig te hebben omdat het probleem zich onverwacht makkelijk liet versimpelen. Wanneer ik mijn plaatje via de menu–interface van R wegschreef werd het een factor 10 kleiner dan wanneer ik het volledig via de programmeertaal liet genereren; Het waarom hierachter ontgaat me (nog steeds). De conversie van EPS naar PDF was toen natuurlijk zo geregeld en had ik een werkbaar plaatje mét fonts in mijn proefschrift staan.

Conclusies

Uiteindelijk is het maken van mijn proefschrift een behoorlijk heftig leerproces geweest en dan niet alleen op TeXnisch en wetenschappelijk gebied, maar ook op

andere aspecten. Het TeX–en was op momenten wel het leukste om mee bezig te zijn, al heb ik er uiteindelijk toch weer minder tijd aan kunnen besteden dan ik me aan het begin van het project had voorgenomen.

Diegene die het eindresultaat nog niet voor zijn geestesoog kan voorstellen wil ik graag doorverwijzen naar mijn persoonlijke website [8].

Referenties

[1] Wilson, P. (2005). *The Memoir Class for Configurable Typesetting*. Available from CTAN via `macros/latex/contrib/memoir/`.

[2] Meijer, E. (2005). *The apacite package*. Available from CTAN via `macros/latex/contrib/apacite/`.

[3] Avadhanula, S., Machowski, M. & Fisher, B. (2005). *The Vim–LaTeX suite*. Available via `http://vim-latex.sourceforge.net/`.

[4] Tantau, T. (2005). *The PGF package*. Available via `https://sourceforge.net/project/showfiles.php?group_id=92412`.

[5] Campbell, C. (2005). *Engspchk Spelling Checker*. Available via `http://www.vim.org/scripts/script.php?script_id=195`.

[6] `http://help-csli.stanford.edu/tex/latex-footnotes.shtml`.

[7] `http://listserv.surfnet.nl/scripts/wa.exe?A2=ind0501&L=tex-nl&F=&S=&P=2474`.

[8] `http://www.paul-lemmens.nl/maps`.

[9] R Development Core Team (2006). *R: A language and environment for statistical computing*. `http://www.r-project.org/`.

Voetnoten

1. Sinds versie 6.0 heeft Acrobat, de volledige suite, de mogelijkheid om PDF–bestanden van commentaar te voorzien en tot op zekere hoogte te bewerken. In hoeverre de niet–commerciële producten deze mogelijkheid ook (gaan) bieden is mij onbekend.
2. De recentste versie van Vim, versie 7.0, heeft een ingebouwde spellingchecker waarvoor nederlandse woordenbestanden eenvoudig toegevoegd kunnen worden.
3. Om de opmaak van dit artikel enigszins leesbaar te houden heb ik veel LaTeX–commando's handmatig afgebroken zodat ze in de kolombreedte passen. Regeleindes gemarkeerd met twee procenttekens behoren op één regel.
4. Hyperref heb ik overigens pas na het goedkeuren en drukken van het proefschrift toegevoegd, om te voorkomen dat de boxen en toevoegingen die dit pakket realiseert (toch) onverhoopt in de gedrukte versie zouden belanden. In hoeverre deze angst gegrond is weet ik echter niet.
5. De Mapsredactie wees me op de optie `pdfpagelayout=TwoColumnRight`, die de 2–page facing optie van Acrobat kan aanzetten.
6. Alle drukkers werken met grote vellen waarop 8 pagina's op de voorkant en 8 pagina's op de achterkant gedrukt worden. Deze vellen worden vervolgens automatisch gevouwen en gesneden tot "die kleine miniboekjes" (katernen) die je in elk gedrukt boekwerk tegenkomt.
7. De redactie van de Maps wees mij op het feit dat pxfonts een volledige vervanger is van palatino, waarbij (ook nog) de cijfers in mathmode door een palatino variant vervangen worden.
8. Vermoedelijk zijn er ook andere cq. betere oplossingen te vinden, maar daarvoor had ik op dat moment geen tijd over.

Paul Lemmens
`paul (at) paul-lemmens (dot) nl`

2.1 Introduction

Traditionally cognitive science and experimental psychology have investigated information processing within individuals (e.g. see Donders, 1868/1969; Kornblum et al., 1990; Sternberg, 1969). Since recently, however, the research investigating interindividual information–processing is gaining momentum (e.g. see Frith & Wolpert, 2003; Van Schie, Mars, Coles, & Bekkering, 2004)**(1)**. Interindividual communication and information processing has been one of the pillars of human–factors research to improve human–computer interaction by trying to make it more like human–human interaction, for instance by using natural–language dialogue. Although often ignored, an essential component of human–human interaction is emotion: A wealth of non–verbal information can be conveyed by gestures or prosody. Even without verbal communication, humans often feel an immediate like or dislike for a person they meet for the first time, or, even between life–long partners communication can ignite a fierce debate with only a few sentences carrying an unexpected and (usually) unintended emotional charge. The success of emoticons in human–computer–human communication (email, chatting and online–presence software) corroborates the important role of emotion in human–human and human–computer interaction.

(1) Check if I have somewhat less neuroscientific literature on this topic.

Originally human–computer interaction (HCI) did not involve emotion at all; the computer was seen as a binary machine devoid of any possibilities to position its responses between the extremes of zero and one and users simply had to adapt to that situation (Brave & Nass, 2003). A similar perspective was taken in experimental psychology where emotion (hot cognition) and reason (cold cognition) were long taken to be completely separated (e.g., Sorrentino & Higgins, 1986). Picard (1997), therefore, argued that perception and appropriate presentation of affect by a computer might facilitate interaction with users and improve user performance in, as well as user satisfaction of, such an interface. She proposed to investigate Affective Computing which she defined as 'all computing that relates to, arises from, or deliberately influences emotions'.

Perhaps one of the reasons why hot and cold cognition have been viewed as separate is that the former is usually associated with responses on the level of the autonomous nervous system (e.g. heart rate, blood–volume pressure, skin conductivity, Brave & Nass, 2003) whereas the latter is associated with higher cognitive mechanisms. Most experiments

Figuur 1. Voorbeeld van een margenoot

Chapter 2

Stimulus–Response Compatibility and Affective Computing: A Review†

I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind.

Lord Kelvin (1824–1907)

Figuur 2. Voorbeeld van een hoofdstuk-pagina

Random bit generator in TEX

Abstract
Een in TEX gecodeerde randomgenerator maakt het mogelijk om willekeurige getallen en beslissingen te verwerken in de productie van documenten.

Keywords
random, pseudorandom, schuifregister, lfsr

Inleiding

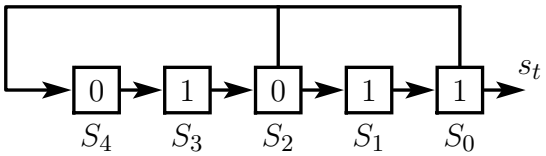
De *hvdm-rng* module maakt bitreeksen voor willekeurige getallen en willekeurige beslissingen. Een voorbeeld waar dit gebruikt kan worden is het zetten van meerkeuzevragen. Door de volgorde van de items te veranderen kunnen dezelfde vragen er van keer tot keer verschillend uitzien. In feite is dit de oorspronkelijke motivatie voor het implementeren van de generator.

Deze module is een op ConTEXt aangepaste en opgewaardeerde versie van mijn vroegere LaTEX *random.sty* package; de laatste versie 2.4 daarvan is gedateerd 1994/10/21 en gepubliceerd in TUGboat.[1] In de hier gepresenteerde versie is de periode van de rij aanzienlijk langer geworden, de macros zijn aangepast aan de manier waarop in ConTEXt parameters worden gebruikt en een paar macros die het gebruik vergemakkelijken zijn toegevoegd. De beschrijving van het generatormechanisme volgt in grote lijnen het TUGboat–artikel. Het wordt hier in het Nederlands gepubliceerd in de hoop dat het voor de lezers van de MAPS nuttig en interessant zal blijken.

Schuifregisters

Omdat een deel van mijn interesses ligt in het lesgeven over geheimschrift in het vak Cryptografie, is het niet verwonderlijk dat daarin de inspiratie voor de macros is gezocht. Willekeurige bitrijen worden in de cryptografie gebruikt om gegevensstromen te vercijferen. Leek dat in 1994 nog een wat exotische toepassing, sindsdien draagt iedereen in z'n mobiele telefoon zoiets bij zich. Van de mechanismen die in aanmerking komen is het schuifregister erg geschikt door zijn eenvoud. En hoewel de eenvoudigste variant, het lineaire schuifregister, uit cryptografisch oogpunt onvoldoende veiligheid te bieden heeft, is het voor toepassing in TEX voldoende zolang beveiliging niet het doel is.

De primaire referentie voor schuifregisters is het befaamde boek van Golomb.[2] Maar er zijn ook meer recente bronnen, waaronder mijn eigen syllabus voor het onderwijs aan de Universiteit van Amsterdam. Voor het begrijpen van de TEX–code is het nuttig een idee te hebben van de werking. De figuur toont een klein lineair schuifregister. Het bestaat uit vijf zogenaamde *registertrappen* die ieder een enkel bit kunnen bevatten. Deze registertrappen zijn van links naar rechts S_4, ..., S_0 genoemd. In de figuur is als voorbeeld in elke trap een bitwaarde ingetekend. Tezamen vormen deze waarden de *toestand* van het register op dat moment; in de figuur is de toestand dus 01011.



Lineair schuifregister met vijf trappen

Het schuifregister werkt als volgt. In elke cyclus wordt het bit in S_0 als resultaat uitgevoerd; op tijdstip t wordt dat het bit s_t. Een nieuw bit wordt gemaakt door optelling van de bits in S_2 en S_0; dat zijn de registertrappen van waaruit een verticale lijn naar boven loopt. Deze optelling geschiedt onder verwaarlozing van de overloop en de mogelijke uitkomsten zijn dus: $0+0 = 1+1 = 0$ of $0+1 = 1+0 = 1$. Het nieuwe bit neemt plaats in S_4. Op hetzelfde moment schuiven alle bits een plaats op naar rechts: $S_4 \rightarrow S_3$, $S_3 \rightarrow S_2$, enz. In het voorbeeld zal $s_t = 1$ en de nieuwe toestand wordt 10101.

Men kan zo vaak een nieuw bit produceren als nodig. Uiteraard zal de reeks zich op een gegeven moment gaan herhalen. Als het register om te beginnen 00000 bevat is dat meteen al het geval. Gemakkelijk valt in te zien dat het nieuwe bit dan ook weer een 0 wordt en dat blijft zo. Hiervan kan gebruik gemaakt worden om de randomproductie effectief uit te schakelen zonder iets aan het programma te behoeven te veranderen. Indien de constructie van het schuifregister handig gekozen is leidt elke andere begintoestand tot een nieuwe, totdat alle mogelijkheden zijn uitgeput. Voor een n–traps register zullen dit er $2^n - 1$ zijn; 31 in het voorbeeld.

Het schuifregister kan worden beschreven met een polynomiale functie in de bit variable $x \in \{0, 1\}$. Deze functie, de zogenaamde *karakteristieke functie* luidt voor het voorbeeld

$$f(x) = 1 + x^2 + x^5$$

Wanneer de karakteristieke functie aan bepaalde eisen voldoet is dat een garantie voor de maximaal mogelijke periode. Een tweede eis die aan het schuifregister werd opgelegd is, dat het eenvoudig te implementeren moet zijn. Dit houdt onder meer in dat het register hooguit 31 trappen mag tellen, omdat anders de toestand niet meer in een countregister past en overflow geeft. Hoe minder aftappunten (S_2 en S_0 in het voorbeeld), hoe eenvoudiger de implementatie; twee stuks geeft een karakteristieke functie met drie termen en is het theoretisch minimum. Een geschikte polynoom die aan al deze eisen voldoet is die welke in het aangehaalde TUGboat–artikel gebruikt is

$$f(x) = 1 + x^{21} + x^{22}$$

Deze levert een random rij met een periode van 4.194.303 bits en is eenvoudig te implementeren zonder dat een hulpregister nodig is. Een andere goede mogelijkheid is het schuifregister gebaseerd op

$$f(x) = 1 + x^2 + x^{29}$$

Dit register heeft een flink langere periode van 536.870.911 maar gebruikt een hulpregister bij het berekenen van het volgend bit.[3] In tegenstelling tot de versie van 1994 is nu gekozen voor de langere reeks.

Implementatie

De toestand van het schuifregister wordt opgeslagen in countregister `\SRstate`, de constante `\SRconst` heeft de waarde 2^{28} en wordt gebruikt om door bijtelling het hoogste bit een 1 te kunnen maken, `\SRlast` dient voor het bewaren van het laatst gemaakte getal en `\SRtemp` is een hulpregister.

```
\newcount\SRstate
\def\SRconst{268435456}
\newcount\SRlast
\newcount\SRtemp
```

Het effect van een stap op het schuifregister wordt bereikt door het register te delen door twee, als gevolg waarvan alle bits een plaats naar rechts opschuiven. Het oorspronkelijk minst significante bit wordt vergeleken met dat twee plaatsen links ervan. Indien deze twee van elkaar verschillen moet een 1 aan de linkerkant van het statusregister worden toegevoegd. Macro `\SRadvance` bewerkstelligt dit. De veranderingen in het statusregister geschieden globaal, zodat het schuifregister onder alle omstandigheden blijft doorlopen en niet bij het verlaten van een lokale groep terugspringt naar een vorig punt in de cyclus. Dit toch al noodzakelijk globale karakter van de update maakt het aantrekkelijk om met een `\begingroup`..`\endgroup` paar de verandering van het hulpregister lokaal te houden.

```
\def\SRadvance{\begingroup
  \SRtemp\SRstate
  \divide\SRtemp\plusfour
  \ifodd\SRstate
    \global\divide\SRstate\plustwo
    \ifodd\SRtemp
    \else
      \global\advance\SRstate\SRconst\relax
    \fi
  \else
    \global\divide\SRstate\plustwo
    \ifodd\SRtemp
       \global\advance\SRstate\SRconst\relax
    \fi
  \fi\endgroup}
```

Deze macro is de kern van de generator. De andere macros zijn voor het gebruik ervan.

`\SRset`, `\SRset[n]`

Het schuifregister blijft stationair in de eerste vorm of als expliciet n = 0 en levert dan alleen nullen af. Voor alle andere input wordt de begintoestand ingesteld op een waarde beneden het maximum van 2^{29}. Extra code is toegevoegd om rekening te houden met de mogelijkheid dat voor de initialisatie een klein getal is opgegeven. Bij initialiseren met een klein getal worden in het begin lange reeksen nullen geproduceerd. Dit komt omdat in de 29 trappen van het register dan veel nullen op een rij zitten. Op zich is dat niet erg en theoretisch moet zo'n reeks altijd een keer voorkomen. Maar het maakt dat het begin van de randomrij er een beetje weinig random uitziet. Vooral indien maar weinig randomgetallen nodig zijn is dat een nadeel. Daarom doet het register eerst 1000 stappen om op gang te komen.[4] Met de huidige snelle computers zal het amper merkbaar zijn.

```
\def\SRset{\dosingleargument\doSRset}
\def\doSRset[#1]{%
  \doifemptyelse{#1}%
    {\global\SRstate\zeropoint}%
    {\global\SRstate#1\relax}%
  \ifnum\SRstate<\zeropoint
    \global\SRstate-\SRstate
  \fi
  \SRtemp\SRconst\relax
  \advance\SRtemp\SRtemp
```

```
  \advance\SRtemp\minusone
  \loop \ifnum\SRstate>\SRtemp
    \global\divide\SRstate\plustwo
  \repeat
  \SRtemp\plusthousand
  \loop \ifnum\SRtemp>\zeropoint
    \advance\SRtemp\minusone
    \SRadvance
  \repeat
  \ignorespaces}
```

`\SRbit`
Genereert het eerstvolgende randombit.

```
\def\SRbit
   {\SRadvance\ifodd\SRstate 1\else 0\fi}
```

`\ifSR` <true> `\else` <false> `\fi`
Neemt de true–tak als het schuifregister een 1 afgeeft en de false–tak voor een 0.

```
\def\ififtrue{\iftrue}
\def\ififfalse{\iffalse}
\def\ifSR{\SRadvance
    \ifodd\SRstate \expandafter\ififtrue
    \else          \expandafter\ififfalse
    \fi}
```

`\SRnumber[n]`, `\SRnext[n]`
Macro `\SRnumber[n]` maakt een getal van n bits. Dit getal wordt geproduceerd door `\SRnext[n]`, dat het in register `\SRlast` plaatst. Deze splitsing over twee macros is gedaan omdat men niet eenvoudig het resultaat van `\SRnumber` aan een countregister kan toekennen. Eerst produceren in `\SRlast` en vandaaruit toekennen aan het gewenste register lukt wel.

```
\def\SRnext[#1]{\SRtemp#1\relax
    \global\SRlast\zeropoint
    \loop \ifnum\SRtemp>\zeropoint
        \advance\SRtemp\minusone
        \global\advance\SRlast\SRlast
        \SRadvance
        \ifodd\SRstate
            \global\advance\SRlast\plusone
        \fi
    \repeat
    \ignorespaces}
\def\SRnumber[#1]{\SRnext[#1]\the\SRlast}
```

Notes

1. H. van der Meer, *Random Bit Generator in TEX*, TUGboat vol.15 (1994) nr.1, p. 57–58.
2. S.W. Golomb, *Shift Register Sequences*, 1967, Holden Day, San Francisco.
3. Strikt genomen is deze bewering onjuist. Het is mogelijk om de berekening zo uit te splitsen dat een hulpregister niet nodig is. Maar waarom gekunsteld programmeren als dat niet echt nodig is?
4. Dit is analoog aan hetgeen in een mobiele telefoon gebeurt bij het instellen van de schuifregisters voor elk spraakframe; deze doen 100 stappen voordat de uitvoer gebruikt wordt.

Hans van der Meer
hansm (at) science (dot) uva (dot) nl

Enjoy TEX pearls diving!

Pearls Session – BachoTEX 2006

The BachoTEX 2006 conference continued the Pearls of TEX Programming open session introduced in 2005 during which volunteers present TEX-related tricks and shorties.

What was requested :

- short TEX, Metafont or MetaPost macro/macros (half A4 page or half a screen at most),
- the code should be generic; potentially understandable by plain-oriented users,
- results need not be useful or serious, but language-specific, tricky, preferably non-obvious,
- obscure oddities, weird TEX behaviour, dirty and risky tricks and traps are also welcome,
- the code should be explainable in a couple of minutes.

Already collected pearls can be found at http://www.gust.org.pl/pearls. All pearl-divers and pearl-growers are kindly asked to send the pearl-candidates to pearls@gust.org.pl, where Paweł Jackowski, our pearl-collector, is waiting impatiently. The pearls market-place is active during the entire year, not just before the annual BachoTEX Conference.

Note: The person submitting pearl proposals and/or participating pearls session needs not necessarily be the inventor. Well known hits are also welcome, unless already presented at one of our sessions.

If you yourself have something that fits the bill, please consider. If you know somebody's work that does, please let us know, we will contact the person. We await your contributions even if you are unable to attend the conference. In such a case you are free either to elect one of the participants to present your work or "leave the proof to the gentle reader" (cf. e.g. http://www.aurora.edu/mathematics/bhaskara.htm).

Is there a font? – *Frank Mittelbach*

The following code allows to check, if some font (tfm metric) is available.

```
\batchmode
\font\X=<font name>%
\errorstopmode
 \ifx\X\nullfont
   <font not available>
 \else
   <use the font>
 \fi
```

Having ε-TEX we can switch between modes using handy \interactionmode primitive.

\relax ex machina – *Paweł Jackowski*

Guess what is the meaning of macros in the following cases:

```
\edef\stra{\csname undefined\endcsname}
\edef\strb{\ifnum0=1\else\fi}
\edef\strc{\ifnum0=0\else\fi}
\edef\strd{\relax}

\meaning\stra % -> \undefined (plus side effect alike
              %      \let\undefined\relax)
\meaning\strb % -> (empty)
\meaning\strc % -> \relax
\meaning\strd % -> \relax
```

...and can you explain why `\ifx\strc\strd` is false, although both control sequences have the same meaning?

```
\ifx\strc\strd true\else false\fi \message{?\strc?\strd?}
```

Since the behavior of is somewhat weird, I've learned to dislike the acting as the universal string delimiter in cases such as

```
\def\gobbler#1\relax{}
\expandafter\gobbler\strc whatever \relax

\def\iterator#1{%
  \ifx\relax#1\else\message{Can you see that #1?!}%
  \expandafter\iterator\fi}
\expandafter \iterator \stra whatever \relax
```

Instead I propose

```
\def\endstr{\noexpand\endstr}
```

– no endless loop in spite of recursion (expands to itself)

– alike `\relax`, `\endstr` returns nothing (a sort of...) if typeset

– side effect is that `\endstr` stops assignments (as `\relax` do)

– `\ifx\endstr\relax` is false, so we can distinguish them

– but be careful: `\if\endstr\relax` is true

\relax ex machina II – *Bernd Raichle*

Question: Is the following TEX input correct, i.e, will TEX abort with an error message or not?

```
\hbox \relax \relax {haha}
\hbox to \hsize \relax \relax {hehe}
\moveright .25\hsize \relax \relax \hbox{hihi}
```

```
\toks0 = \relax \relax {hoho}
\toksdef\tlist = 2  \tlist = \relax \relax {huhu}

\setbox0 = \relax \hbox \relax {haha}

\global \relax \global \relax \long \relax \def\foo{oh!}

$$ \halign{\kern 20pt #\cr
           \noalign \relax \relax {aha}\cr
           oho\cr} \relax $$

$ \left\relax( 1\mathord\relax {+} 2^\relax{3} \right\relax)$

a\leaders \hbox \relax {\TeX} \relax \hskip .5\hsize b

bla \vadjust \relax \relax {foo} bla.

B\accent 127\relax \relax \relax \relax a%
\discretionary \relax {k-} \relax {k} \relax {ck}%
er.
```

Answer: Every token above is not needed. Nonetheless these tokens do not cause any error, they are allowed and ignored at the shown places.

If you browse through the TEX source code file `tex.web` you will propably have seen the web chunk named `@<Get the next non-blank non-relax non-call token@>`. This web chunk is used in the procedure `scan_left_brace`, in `box_end` for leaders, in `scan_box`, in `scan_math`, in `scan_delimiter`, in `prefixed_command`, in assignments of `\toks` or `\toksdef` tokens after the equal sign, in `do_assignments`. These procedures are used to scan the input for some TEX primitive constructs. The effect is that spaces and `\relax` tokens can be inserted at various places without harm because they are ignored.

Read only first line of an input file – *Bernd Raichle*

You have a TEX input file with more than one line:

```
test file 1 -- line 1 \count0=42
test file 1 -- line 2 \count0=31415
test file 1 -- line 3 \count0=1
... more lines ...
```

You can use the following trick, if you want to process only the first line of this file:

```
\endinput \input testfile
```

This will produce the following output:

```
test file 1 -- line 1
```

Background: The primitive `\endinput` does not close the current file immediately. It sets a TEX internal flag that indicates that if the next end-of-line is

processed the current file is closed. Thus all the tokens after the \endinput until the end of the line are processed. If you start reading a new file after \endinput this file will become the current file and it will be closed after the first end of line is seen.

```
\immediate\openout0=\jobname.out
\immediate\write0{test file 1 -- line 1 \count0=42}
\immediate\write0{test file 1 -- line 2 \count0=31415}
\immediate\write0{test file 1 -- line 3 \count0=1}
\immediate\closeout0
...
\endinput\input zz.out (\the\count0) \end
```

As a result we get:

... test file 1 – line 1 (42)

\input file inside a macro definition – *Bernd Raichle*

If you have a file and want to read its contents in a macro definition, TEX does not allow to simply use

```
\edef\foo{\input file }
```

because at the end of a file TEX checks if you are inside an incomplete \if statement or \def macro definition and outputs the error message

```
! File ended while scanning text/definition of ...
```

Nonetheless there is a non-trivial way to avoid this error message:

```
\immediate\openout15=bt-pp-r4.tex
\immediate\write15{Another TeX}
\immediate\write15{Pearl\string\noexpand}
\immediate\closeout15

\edef\foo{Yet \input bt-pp-r4.tex !}
\message{\meaning\foo}
```

The primitive \noexpand does all the magic and you will get

```
macro:->Yet Another TeX Pearl!
```

without any error message.

Question: Why has \noexpand this effect?

If you are using ε-TEX, you can set its new special token list \everyeof to \noexpand to achieve this effect without changing the file.

4 endless lines – *Taco Hoekwater*

The end of a line in an \input-ed file normally creates a space in the output, TEX appends a character with the current value of \endlinechar to each that character is later converted to a space.

```
\immediate\write18
  {echo -n Y >bla.tex}
X\input bla Z
% gives XY Z
```

If you do not want that space, for instance because you want to typeset the of the file in-line, then there are a number of options. Two are usable writing, and two others during type reading.

1. Writing a percent sign to the end of the line works

```
\immediate\write18
  {echo -n Y\letterpercent >bla.tex}
X\input bla Z
% gives XYZ
```

2. Another option is ending the written line with \relax or a similar space-gobbling command.

```
\immediate\write18
  {echo -n Y\letterbackslash\letterbackslash relax >bla.tex}
X\input bla Z
% gives XY\relax Z
```

3. Temporarily setting \endlinechar to an impossible value like -1 is a possibility

```
\immediate\write18
  {echo -n Y >bla.tex}
X{\endlinechar=-1 \input bla }Z
% gives XYZ
```

4. Changing the catcode of the current \endlinechar to 9 (ignored) also works

```
\immediate\write18
  {echo -n Y >bla.tex}
X{\catcode'\^^M=9 \input bla }Z
% gives XYZ
```

Check if defined, no side effects – *Bernd Raichle*

LaTEX and other macro packages include a test if a control sequence is already defined using
\csname...\endcsname. In principle the following definition is used:

```
\def\@ifundefined#1#2#3{%
  \expandafter\ifx\csname #1\endcsname\relax#2\else#3\fi}
```

The use of `\csname...\endcsname` has the side effect that an undefined control sequence will be defined as `\relax` and a simple test using `\ifx<control sequence>\undefined` will fail.

To avoid this side effect, you can change the definition above introducing a group "around" `\csname`:

```
\def\@ifundefined#1#2#3{%
  \begingroup \expandafter\expandafter\expandafter \endgroup
  \expandafter\ifx\csname #1\endcsname\relax#2\else#3\fi}
```

Note: This definition is not fully expandable, thus it will fail in expansion-only contexts where the original definition will work.

If you are using ε-TEX, the new `\ifcsname...\endcsname` primitive can be used instead.

During the presentation of this pearl at the BachoTeX 2006 pearls session, two alternatives to this trick has been proposed. Worthy to note, that the macro above use more `\expandafter`'s then actually needed. Once we say

```
\def\@ifundefined#1#2#3{%
  \begingroup \expandafter
  \ifx\csname#1\endcsname\relax\endgroup#2\else\endgroup#3\fi}
```

we get exactly the same effect in a bit more efficient way. But there is one more sticky problem in all the constructions above; every control sequence defined as `\relax` (and `\relax` itself) is treated as undefined. The solution seems to be

```
\def\@ifundefined#1#2#3{%
  \begingroup \expandafter \endgroup \expandafter
  \ifx\csname#1\endcsname\undefined#2\else#3\fi}
```

Here we expand `\csname...\endcsname` before `\endgroup`, but the condition is fixed outside the group. Thus, `\undefined` instead of `\relax` can be used for comparison.

Global assignments done locally – *Bernd Raichle*

Sometimes it is necessary to define a macro under a changed input regime, e.g, using different category codes or another end line character. Usually this is done inside a group to keep the changes locally and the macro or the token is defined `\global`'ly.

In `plain.tex` you can find the following examples:

```
\catcode`\@=11
```

1. Helper macro for `\newif`:

```
{\uccode`1=`i \uccode`2=`f
 \uppercase{\gdef\if@12{}}} % `if' is required
```

2. Definitions of `\obeylines` and `\obeyspaces`:

```
{\catcode`\^^M=\active % these lines must end with %
  \gdef\obeylines{\catcode`\^^M\active \let^^M\par}%
  \global\let^^M\par} % this is in case ^^M appears in a \write
{\obeyspaces\global\let =\space}
```

3. Definition of \getf@ctor:

```
{\catcode`p=12 \catcode`t=12 \gdef\\#1pt{#1}} \let\getf@ctor=\\
```

4. Math definitions for primes and underscore:

```
% _ in math is either subscript or \_
{\catcode`\'=\active \gdef'{^\bgroup\prim@s}}
{\catcode`\_=\active \global\let_=\_}
```

By placing the begin and end of group tokens in a bit "unusual" way using a temporary token register assignment or a macro definition, all these assignments can be done locally:

1. Helper macro for \newif:

```
\begingroup \uccode`1=`i \uccode`2=`f
\uppercase{\endgroup\def\if@12{}}
% `i'+`f' as delimited arguments are required
```

2. Definitions of \obeylines and \obeyspaces:

```
\begingroup \endlinechar=-1 \catcode`\^^M=\active
  \toks0={\endgroup
    \def\obeylines{\catcode`\^^M\active \let^^M\par}%
    \let^^M=\par  % this is in case ^^M appears in a \write
  }\the\toks0\relax
\begingroup \obeyspaces\def\x{\endgroup\let =\space}\x
```

3. Definition of \getf@ctor:

```
\begingroup \catcode`P=12 \catcode`T=12
  % \lccode`P=`p \lccode`T=`t % is default setting
  \lowercase{\endgroup\def\getf@ctor#1PT{#1}}
  % `p'+`t' with catcode `other'
```

4. Math definitions for primes and underscore:

```
\begingroup \catcode`\'=\active
  \def\x{\endgroup\def'{^\bgroup\prim@s}}\x
\begingroup \catcode`\_=\active
  \def\x{\endgroup \let_=\_}\x
  % _ in math is either subscript or \_
```

There is no advantage of a local definition for the shown plain.tex cases but if you want to do similar definitions within a group, the shown technique can be very helpful.

Note: If you want to define macros with arguments it is better to use a token register assignment because you have to double the hash mark as macro parameter character inside the macro definition text.

\lccode builds a hash table – *Hans Hagen*

Wybo Dekker (NTG) asked on the TEX-NL list how to construct a macro where in the name a '1' would be replaced by an 'A'. This involves a conversion and TEX does not ship with that many straightforward conversion features. His approach involved the \char primitive but failed for the (maybe not that obvious) reason that this primitive results in a character node and in expansions (as in an \edef) remains as it is: a reference to a specific character (slot) in the font used at the moment when it is applied.

```
\Name{test}{1}{Hello}
\Name{test}{2}{World!}
\testA\ \testB % this should give: Hello World!
```

A rather convenient way to remap characters is using the \lccode or \uccode primitives in combination with \lowercase or \uppercase. The following solution is a bit optimized in the sense that we expand the temporary variable before we end the group. An alternative is to define \temp globally and to omit the first \expandafter. Watch how we apply \lowercase to the whole definition; using the \lowercase inside an \edef would not work. Instead of using a loop an dsome calculations, we directly assign the codes. The 0/J case is an exception anyway.

```
\long\def\Name#1#2#3%
 {\bgroup
  \lccode‘1=‘A \lccode‘6=‘F
  \lccode‘2=‘B \lccode‘7=‘G
  \lccode‘3=‘C \lccode‘8=‘H
  \lccode‘4=‘D \lccode‘9=‘I
  \lccode‘5=‘E \lccode‘0=‘J
  \lowercase{\def\temp{#2}}%
  \expandafter\egroup\expandafter\def\csname#1\temp\endcsname
    {#3}}

\Name{test}{1}{Hello}
\Name{test}{2}{World!}
\testA\ \testB
```

Of course we don't need such a remapping at all, which is demonstrated by the alternative posted by Piet van Oostrum at TEX-NL:

```
\long\def\Name#1#2#3%
 {\expandafter\def\csname #1\ifcase#2\relax\or A\or B\or C\or
    D\or E\or F\or G\or H\or I\or J\or K\fi\endcsname{#3}}
```

A better variant is the following. This one takes care of the zero case:

```
\long\def\Name#1#2#3%
 {\expandafter\def\csname #1\ifcase#2 J\or A\or B\or C\or D\or
    E\or F\or G\or H\or I\else#2\fi\endcsname{#3}}
```

The next one is even better because it also takes care of non digits:

```
\long\def\Name#1#2#3%
 {\expandafter\def\csname#1\ifx#20J\else\ifcase0#2\or A\or B\or
    C\or D\or E\or F\or G\or H\or I\else#2\fi\fi\endcsname{#3}}

Now we can say:

\Name{test}{0}{zero}
\Name{test}{1}{one}
\Name{test}{x}{xxx}
\testJ \testA \testx
```

Sigma tweak – *Bogusław Jackowski & Piotr Strzelczyk*

A sum operator that adjusts its width automatically to the widest subscript or superscript – perhaps completely useless ;-)

Note: pdfTEX is required for clippling; you may wish to use other tools for this purpose

defining the parts of the symbol:

```
\def\SIGMAleft{\copy0 }
\def\SIGMAright{\copy1 }
\def\SIGMAcenter{\copy2 }
```

preparing the parts of the symbol:

```
\setbox0\hbox{\tenex \char88\kern-.6em}
\setbox1\hbox{\kern-.8em\tenex \char88}
\setbox2\hbox{\hbox{\kern-.8em\tenex \char88\kern-.6em}}
```

clipping:

```
\pdfxform0 \setbox0\hbox{\pdfrefxform\pdflastxform}
\pdfxform1 \setbox1\hbox{\pdfrefxform\pdflastxform}
\pdfxform2 \setbox2\hbox{\pdfrefxform\pdflastxform}
```

assembling:

```
\def\SIGMA{%
  \mathop{\hbox{\SIGMAleft \kern-.1em
  \xleaders\SIGMAcenter\hfill
  \kern-.1em\SIGMAright}}\limits}
  % with \nolimits behaves ‘‘traditionally’’
```

usage:

```
$ \SIGMA^{\rm a}\quad
  \SIGMA^{\rm sum}\quad
  \SIGMA^{\rm operator}\quad
  \SIGMA^{\rm that\ adjusts}\quad
```

```
  \SIGMA^{\rm its\ width\ automatically}
        _{\rm to\ the\ widest\ subscript\ or\ superscript}
$
```

As a result we get:

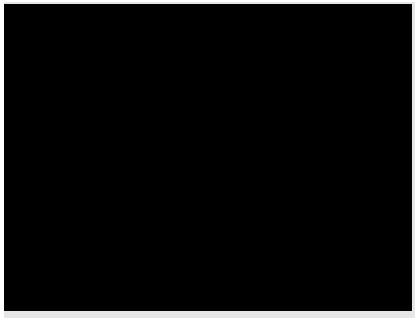
$$\sum_{\text{a}} \quad \sum_{\text{sum}} \quad \sum_{\text{operator}} \quad \sum_{\text{that adjusts}} \quad \sum_{\text{its width automatically}}^{\text{to the widest subscript or superscript}}$$

The final pearl is added by your Maps editors. While Hans and I were at TUG, Paweł became the proud father of *Kasia Jackowska*. In Paweł's words:

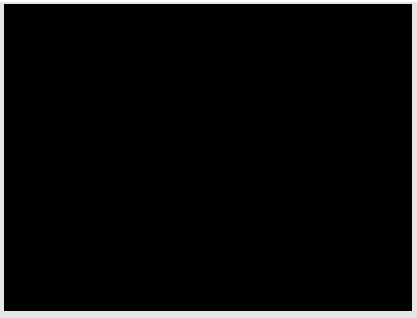
> "7th November at 14:35pm our daughter was born. 3475g, 54cm and needless to say, the most beautiful allovertheworld."

Our congratulations to the proud parents and grandparents!

Deus ex machina – *Paweł Jackowski*



A `\futurelet` Gust member.



An example of `\expandafter`.

Epspdf

Easy conversion between PostScript en Pdf

Abstract
This article introduces epspdf, a converter between eps, PostScript and pdf which can be run either via a graphical interface or from the command-line.

Keywords
Pdf, PostScript, eps, Ruby, Tk, cropping, page selection, Ghostscript, pdftops

Introduction

When preparing a LaTeX document, it is often convenient to have graphics available both in eps- and in pdf format. Epspdf[1] improves on previous solutions by having both a CLI or command-line interface and a GUI, by converting both ways, using pdftops from the xpdf suite[2], and by various new options, which were made possible by round-tripping between PostScript and pdf.

Sample applications

Case 1: Converting a Powerpoint slide to pdf and eps. A.U. Thor writes a paper in LaTeX and creates his illustrations with PowerPoint. He likes to turn these into pdf graphics, so that he can process his paper with pdflatex.

From PowerPoint, he 'prints' to an eps file (see the appendix). The Windows Print dialog is rather insistent on giving the eps file an extension '.prn'. He loads the graphic in epspdftk, where the .prn file is accurately identified as eps. He checks the 'Compute tight boundingbox' option, selects pdf output format, and clicks 'Convert and save'. Some annoying black boxes flit across his screen, but soon a message 'Conversion completed' appears. He presses the 'View' button and Adobe Reader displays what he hoped to see.

He might as well replace the eps with one with a good boundingbox, so he converts the pdf back to eps. It may save epspdf some work if he first unchecks the boundingbox checkbox.

When viewing the resulting eps with GSview, there is once more a lot of whitespace around the figure. Hunting around in the GSview menus, he finds 'EPS Clip' and 'Show Bounding Box' in the Options menu, and with either option checked, he sees that all is well.

Case 2: Converting multiple slides from a PowerPoint presentation to pdf graphics. A.U. Thor, encouraged by his previous success, adds several new graphics to his PowerPoint file. Since epspdftk can handle multi-page documents, he prints the entire document to a PostScript file, which he loads in epspdftk.

He notices that the box with file info doesn't tell him the number of pages. For general PostScript files, there is no sure-fire quick way to get this information.

He checks 'Convert all pages' and sets Output format to pdf. After conversion, the info box does give him the number of pages.

He writes the first page to a pdf file with a tight boundingbox, reloads the complete pdf, then writes the second page to a pdf, and then wonders whether the command-line might not be more convenient.

He reads the epspdf webpage and manual, browses

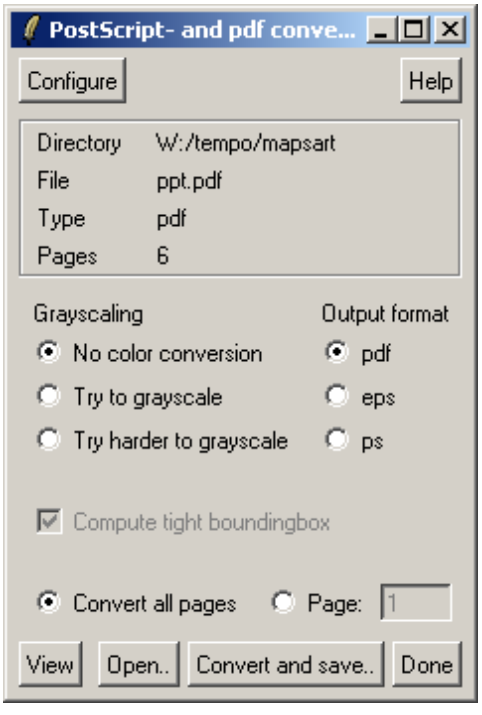


Figure 1. Main window of epspdftk (MS Windows)

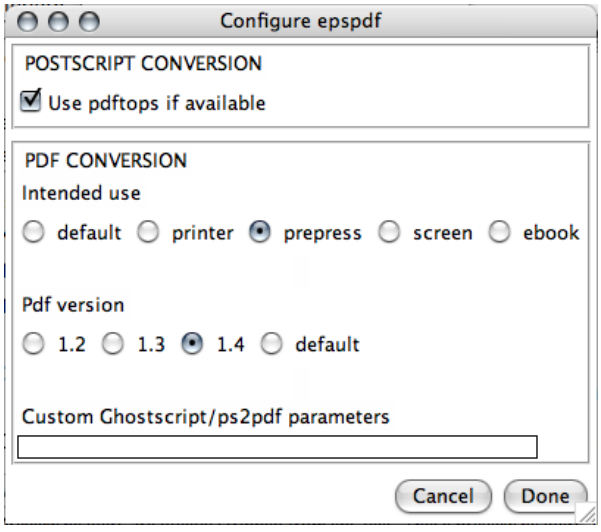


Figure 2. Configuration screen (Mac OS X)

the epspdf directory and comes up with a batchfile epspdf.bat containing the following line:

```
"C:\Program Files\epspdf\bin\ruby.exe"
  "C:\Program Files\epspdf\app\epspdf.rb"
  %1 %2 %3 %4 %5 %6 %7 %8 %9
```

(everything on one line) and types

```
epspdf -b -p 3 ppt_slides.pdf figure3.pdf
```

Then he outdoes himself in cleverness and does the remainder with one command (everything on one line):

```
for %f in (4 5 6) do
  epspdf -b -p %f ppt_slides.pdf figure%f.pdf
```

Case 3: Creating cropped typeset samples. Mrs. TeXHeX writes a paper for the MAPS about her adventures with LaTeX. She wants to display typeset examples with her own fonts and formatting, not with those of the MAPS. So she creates a LaTeX document containing one sample per page, and makes sure, with a preamble command `\pagestyle{empty}`, that each sample is the only content on its page. She compiles the document to a pdf and extracts the samples with a tight boundingbox, in the same way as in the previous example.

Case 4: Embedding fonts into an existing pdf. Ed Itor is collecting papers in pdf format for a conference proceedings. The printshop tells him that one of the submitted pdf files doesn't have all its fonts embedded, which is a no-no, even though the font is just Courier and Ed Itor doesn't quite understand the fuss.

Luckily, when converting PostScript to pdf, Ghostscript can embed standard PostScript fonts (Times etc.) even if they are missing in the PostScript file. Ed Itor goes to the Configure screen and verifies that 'Intended use' is set to to 'prepress'. With this setting, converting to PostScript and back to pdf does the trick.

Warning. It is quite possible that the original document was created with slightly different versions of the missing fonts than the ones included by Ghostscript. Usually this will cause no problems, but one might want to check the result anyway.

Some implementation details

The program is written in Ruby and Ruby/Tk, and uses Ghostscript and pdftops from the xpdf suite for the real work. The installation instructions in the download and on the web page explain how to obtain these programs.

The program consists of several modules. The main ones are a main library epspdf.rb which does double duty as command-line program, and a Ruby/Tk GUI main program. Conversions are organized as a series of single-step conversion methods and an any-to-any conversion method which strings together whatever single-step methods are required to get from A to B.

I have included all conversions and options into the program that easily fit into this scheme.

Configuration. Epspdf saves some conversion options between sessions. Under Windows it uses the registry, under Unix/Linux/Mac OS X a file .epspdfrc in the user's home directory. Besides some precooked options, advanced users can also enter custom options for Ghostscript (GUI and CLI) and for pdftops (CLI only).

The Windows setup program. For Windows, epspdftk is available as a Windows setup program. This includes a Ruby/Tk subset, so there is no need for a full Ruby and Tcl/Tk install. This Ruby/Tk subset is adapted from one generated with RubyScript2Exe[3].

But Windows users can also manually install from a zipfile if they already have Ruby and Tcl/Tk.

Mac OS X. Under Mac OS X, epspdf mostly duplicates functionality from the Preview application. However, when faced with problem files it is nice to have an alternative converter. From the epspdf homepage you can download a double-clickable and dockable AppleScript applet for starting up epspdftk.

Appendix: exporting PostScript from Windows programs

Using a printerdriver. Often, the only way to get EPS or PostScript from a Windows program is by 'printing' to a PostScript file.

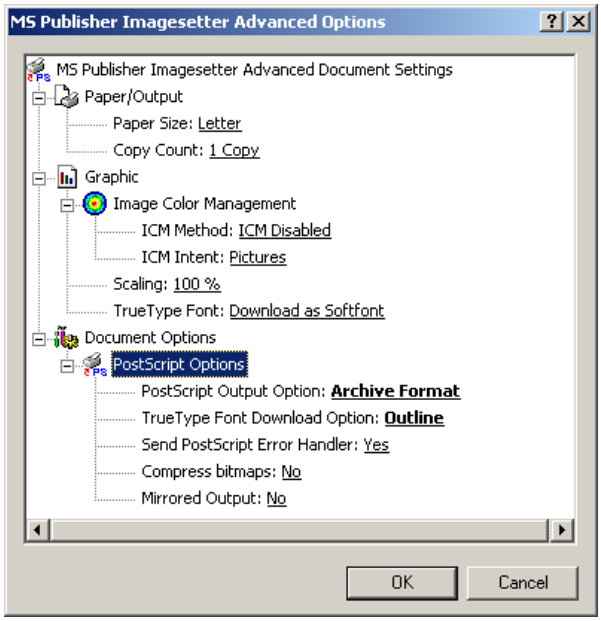


Figure 3. MS Publisher Imagesetter – printer settings

For this, you need to have a PostScript printer driver installed. You can pick 'FILE' for printer port. A suitable driver which comes with Windows is Generic / MS Publisher Imagesetter.

Pay attention to printer settings: in the Print dialog, click 'Properties', then 'Advanced' (on either tab). In the 'Advanced Document Settings' tree, navigate to first 'Document Options' then 'PostScript Options' (figure 3).

For 'PostScript Output Option' the default setting is 'Optimize for speed'. Change that to 'Optimize for Portability' or 'Archive Format', or, for single pages only, 'Encapsulated PostScript'. These non-default options presumably produce cleaner PostScript code, without printer-specific hacks. Experiment with this and other options if you run into problems (e.g. bad-looking screen output, or part of a graphic getting cut off, or conversion to bitmap).

What works best may depend on your Windows version: under Windows 2000, Archive worked best for me, but Taco Hoekwater warns me that this option was unusable in older Windows versions.

Another setting here to pay attention to is 'TrueType Font Downloading Option'. Pick 'Outline', not 'Automatic' or 'Bitmap'.

Using a program. Other possibilities for generating PostScript or pdf are the TpX and wmf2eps programs, which both can read Windows wmf- and emf files and also have options to write clipboard contents to an emf file. Wmf2eps uses its own virtual PostScript printer driver in the background. For faithful conversion, pick wmf2eps; for subsequent editing, choose TpX. Both programs are availbable from CTAN[4].

URLs

1. `http://tex.aanhet.net/epspdf/`
2. `http://www.foolabs.com/xpdf/`
3. `http://rubyforge.org/projects/rubyscript2exe/`
4. `http://www.tug.org/ctan.html`

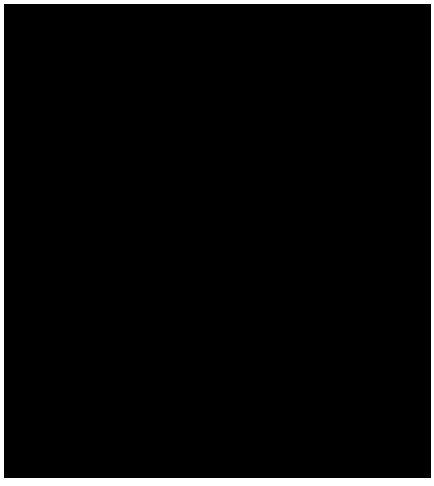
Siep Kroonenberg
`siepo at cybercomm dot nl`

David Walden interview

A conversation about writing and learning and some books to read

Introduction

There's a treasure URL (http://www.tug.org/interviews/) on the TUG site, where Dave Walden has collected a number of excellent interviews with key people of the TEX community – a lively "Who's Who" for anyone who has met some TEX luminaries or seen them in action during conferences. The interviews go way beyond the obvious as Dave invites his guests to respond to his lucid questions. The result is a growing collection of significantly detailed portraits of the people who have made the TEX landscape the way it is today. Even if you've known one of the featured people for years, you're certain to discover something interesting about this person that you've never been aware of. Although there *is* an excellent interview with Dave himself (http://www.tug.org/interviews/interview-files/dave-walden.html) on the site, conducted by Karl Berry, we decided to interview Dave for MAPS, exploring some subjects that were mentioned in his online conversation with Berry. You might want to read that interview first to have context for some of the questions and answers in this interview.



FG: In your career, you've been a dedicated writer of documents, like manuals. As a highly significant witness to great inventions, you've described what was created so well that some people remember you as the inventor, for instance of telnet. And you were nearby when the first computer adventure game Adventure was developed. Are you an inventor, a writer or both?

DW: My primary tasks in my business career were as an actual inventor, as a manager of inventors, and as a manager of businesses involving innovation. I wrote a lot for four reasons: (1) it helped me think things through; (2) I was quick at it and it made sense for me to do it rather than trying to force someone to do it for whom it was a struggle, (3) I wanted to be sure to get my name on the eventual published document, and (4) (as Will Crowther told me when I was in my first job about one month out of college) the person with the pen has disproportionate influence on the outcome of the discussion. This list is not in priority order – I will certainly claim that point 3 was my least concern.

The first point is really about learning, whether I am trying to understand what someone else said or what I myself am thinking. I have always written down my own thoughts when they begin to get complicated (e.g., during a technical design session) because it helps me see flaws or gaps in my reasoning – sort of the same reason one has a written contract about a business deal rather than just a verbal understanding. Also (since college where I barely studied at all) I have taken detailed notes when hearing a presentation or lecture, and I often write-up these notes into nearly a verbatim transcript afterwards to help me sort out what I thought had been said. This is the way I came to be co-author of my first book, (*A New American TQM*). I listened to six full days of lectures by Shoji Shiba and took detailed notes. Then I wrote these up so I could pass the information on to the other people in my company (I was chief quality officer of my company at the time). In the end, I had something that was close to what became the published book. This use of writing is in keeping with a model of skill development that Shoji Shiba promotes: first you are taught something (hear the lecture); then you say it back in your own words (write it down, try to repeat it to other people) with feedback from an expert; then you practice it yourself, repeatedly. At each stage you get greater insight into and proficiency with the skill you are trying to learn. Another reason my "Travel's in TEX Land" column tells about my own experiences is

that by writing them down I understand better what I have done. In fact, I kill several birds with one stone: I understand better myself, I perhaps help someone else understand something, and I fulfill my commitment to provide a regular column to *The PracTeX Journal* without having to learn more than what I actually did.

FG: What are your thoughts about the power of words written by an expert observer?

DW: I don't have much thought on this as my experience was as a participant, not an observer. However, how history is remembered in the long term is how it is written down, whether the writing is accurate or not. [I just read Stephen King's book *On Writing* and in something related to this book his wife (also a writer) is quoted as saying something like "I prefer fiction to non-fiction and I think the latter term is highly inaccurate."]

FG: Which writers / novelists / poets do you read and admire?

DW: I read widely, both fiction and non-fiction, but just what happens to strike my interest, not because of the quality of the writing. I get most of my reading material by looking at what is on the "new book shelf" at my town library, picking out a bunch of books, reading one or two that engage me and returning the rest unread; I make several trips a week to the town library. I don't read poetry, but of course some writers have poetic skill with words. I tend to look for more books by the same writer when I find one book I like by the writer. Of some books that seem particularly well written, I often ask myself "how does he/she do that", but I don't have much insight into how it is done. I can of course list some novels or writers I have enjoyed (and will be happy to do so) but I'm not sure that any list I give would be representative of anything.

What I can say about books and writers is that they can provoke lots of useful thought for me even if the book is not meant explicitly to be "educational" or a "how-to" book. For instance, *Moneyball* by Michael Lewis (a book about the US sport of baseball and the best non-fiction book I think I have read in the last decade) makes one think about the importance of evidence versus "gut feel" and wishful thinking.

Similarly, John Irving's fictional book *The Cider House Rules* gives pause to one's knee-jerk reactions about the controversial issue of abortion. I also think plays and movies can be equally thought provoking and that the writing is key for plays and movies (at least for the movies and plays I enjoy most) just as it is for books despite what set designers, actors, and directors can bring to these efforts. (I try to see a few plays and 100 movies (http://www.walden-family.com/public/movie-index.htm) each year.)

Of course, some books do explicitly try to teach something and some of these can be quite wonderful, or at least quite educational. One of the most valuable books I ever read (studied, actually) was Robert Anthony's *Essentials of Accounting*. Anyone who dismisses double-entry bookkeeping as boring or too complicated has thrown away the possibility of making use of an tremendously powerful organizational tool – that the Medicis created a tool that is practically a miracle of elegance and usefulness is clear from the fact that the method has been in wide use since the 15th century.

FG: Many organizations depending on volunteer work, like TUG and NTG, find that these are harder times than a decade ago. Today it seems more difficult to get work done and conferences get fewer members to attend. You take a different view though. Your thoughts struck me as quite original in the way one can compare the challenge of today's volunteer organizations with for-profit organizations that also have resource conflicts.

DW: My view is that all organizations have many of the same kinds of issues: maintaining a set of "customers" for whatever they do, finding/selecting people who can succeed in getting key work done (just because businesses can pay people doesn't mean they automatically have access to the appropriate people), organizing and engaging people to do multi-person tasks, adapting to changes in the world, remaining financially solvent, etc. Of course, some of the differences are real. By the way, I think "for-profit" is not part of the distinction we are talking about – many non-profit organizations (e.g., Red Cross, Catholic Church, Army) have just as many organizational issues as for-profit businesses do; we are talking about organizations which depend primarily on volunteers versus those which depend primarily on paid employees. It is not even necessarily a matter of priorities: I know of plenty of people who care more about the activities of their volunteer organization (their real life) than they do about the activities of the company they are forced to work for to earn a living. [By the way, the great management thinker Peter Drucker made the point about for-profit businesses that profit is a *cost* of doing business, not the goal – the goal is creating and keeping customers. He discusses this in his monumental book, *Management: Tasks, Responsibilities, Practices*. Drucker also wrote a book on *Managing the Non-profit Organization*.]

Notice that one of the activities volunteer-based organizations have in common with regular businesses is that they must adapt to changes in the world.

The TEX world has significantly changed since TUG was founded. In the early days TEX was part of, or perhaps the instigator of, a state-of-the-art research area, TEX was uniquely capable of doing computer-aided typesetting, and TUG was a key to communication among developers and diffusers of TEX. Today computer typesetting is not the wide-open research area it once was, many alternatives to TEX exist for computer-aided typesetting (some of which, e.g., InDesign, have massive marketing behind them), very mature TEX distributions abound in the world, and the web (e.g., comp.text.tex) has replaced the user groups as the primary way of communicating about TEX. It's no wonder that involvement with the user groups has diminished. I think the user groups still need to do some work to figure out how they can best serve this changed world. Complicating things is the fact that the TEX culture is substantially a "free" or "open-source" culture; this means that the user groups tend to give away publicly the good things they develop which additionally decreases the incentives to participate explicitly in the user groups. Personally, I suspect future vitality for TEX depends on a few people with great capability putting in massive amounts of effort to develop production versions of follow-on systems to TEX, just like Knuth did originally. In other words, TEX or its follow-on needs to be widely perceived as being state-of-the-art again. We may be seeing some of that with the efforts of highly capable and motivated people like Hans Hagen, Taco Hoekwater, Hàn Thế Thành, and Jonathan Kew, to name just a few of the potential heavy hitters that come to mind. Personally I don't think not having funding like private industry does for major projects is the key issue – "enough" funding will be found as the right people make themselves available.

FG: Another subject that intrigues me, and I'd like to explore this subject with you, pops up in several spots in Karl Berry's interview with you: your awareness of non-altruism in volunteer work, and your way of preventing stress in your work in order to make your time all the more productive. You said "I like projects where I also can benefit from what I am doing. I also prefer projects that can be done incrementally so I am not under too much pressure to finish too much too soon; one of my rules of thumb is to avoid trying things that are so hard or have such a near term deadlines that it becomes stressful or an unpleasant burden to work on them."

DW: Regarding altruism, I believe people mostly do what they do because it does something for them, whether they are working for businesses or volunteer organizations. Consciously or unconsciously some people work for money, some people work for glory, some people work because they like to learn, some people like to be a part of a team, some people like to feel they are helping others, some people want to be appreciated, some people like to be martyrs, some people are working toward some sort of immortality, some people do what they do because they are good at it and it feels good to be able to keep doing it, some people want to have impact on the world, some people like a feeling of accomplishment, some people like to feel altruistic, etc. In either type of organization, a key organizational task is finding people who are motivated (for whatever reason) and capable to do what needs to be done. [You might look at the parts of the following paper, http://cqmextra.cqm.org/cqmjournal.nsf/reprints/rp11300, by Steve Kelner that talks about the three motives people typically have (accomplishment, affiliation, and power); Steve has also written a book applying this theory to writing: http://www.upne.com/1-58465-442-2.html]

Personally, I enjoy learning more about something I am interested in, and I also enjoy being able to accomplish something tangible – these are two of my motivations from the list of example motivations I just gave you. Thus, I try to organize my volunteer activities in ways that are compatible with these motivations. I also tried to organize my activities when I was working for pay so I was doing stuff I wanted to do more often than not and had a good chance of succeeding at what I was doing. There are always plenty of things that need to be done in any organization and different people are skilled at different things, so it just makes sense for the workers and the managers to work together to try to match people to what they are good at. [As I remember, Peter Drucker also has something to say about this, e.g., in his brilliant memoir, *Adventures of a Bystander*.]

Of course, one always has to do some work one doesn't like, and it also makes good sense to just try to make short work of that part rather than fighting it so one can get back to what one is good at and enjoys. Doing this matching doesn't require a super high level of awareness about what oneself or others are good at and enjoy, but it does require an honest view of such strengths and weaknesses. Whether as a worker or as a manager, I always want to succeed (no one ever cares about a person's success more than the person himself or herself). As a worker, I know I will ultimately go farther by focusing on what I am good at and want to do than by trying to be someone I am not. As a manager, I know that my success depends on the people working for me succeeding (I still care about my success more than anyone else), and so I must try to match them to what they are good at. There

is a lot of talk about "empowerment" these days: to me empowerment means that someone (1) has the capability to do something, (2) has the authority and responsibility to do that thing, and (3) is engaged by the desire to do that thing. Two of the three are not enough. If we can get everyone empowered, we can get a lot done.

The point about avoiding stress is that when something becomes too much of a burden for someone (i.e., they are failing rather than succeeding at a task), then the person tends to wander away (mentally or physically) to do something that is more fun. This happens for paid employees as well as for volunteers. This is something one has to constantly guard against, as a worker or as a manager, and to take steps to make the job accomplishable or to quickly move the struggling person (before too much time has been wasted) to a more suitable assignment. Complicating such considerations is the fact that some people – a few – are able to reorganize themselves and work their way out of situations in which they are in over their heads, and they need to be allowed to continue, but perhaps with a little guidance so their journey is not too inefficient.

The point about doing what you are good at – for greater success and for greater enjoyment – bears repeating and reminds me of something Edward O. Wilson (the renowned biologist *and* Pulitzer-Prize winning author) suggested in his wonderful autobiography, *Naturalist*. If my memory serves me correctly, he expounded the joys of taxonomy – describing and organizing things – in contrast to making fundamental discoveries. Some people are better at one of these and some people are better at the other, both can make a major contribution to the world, and it pays to figure out which type you are. Wilson talks about skirting your weaknesses and pushing your strengths. Read this book if you haven't already.

FG: In addition to the TUG interviewing work and your PracTeX Journal column, what writing and publishing are you doing now?

DW: A few months ago I finished writing a new book called *Breakthrough Management* (http://www.walden-family.com/breakthrough/) with my co-author Shoji Shiba. This is fundamentally his work, but he doesn't write for publication in English and he and I have worked together teaching and writing about management for long enough now that I was also able to contribute something to the content of the book. Our previous books, *A New American TQM* and *Four Practical Revolutions in Management*, were published by a traditional publisher; with this latest book, however, we decided to experiment with modern printing and distribution technologies that allow authors to control their own publishing and hopefully get a little more profit than is possible with traditional books (at least with traditional books that may only sell a few thousand to 10,000 or 20,000 copies). Thus, in recent months I have spent much time (1) using LaTeX for the first time to actually do the finished typesetting of a whole book, (2) learning about printing options including print-on-demand, and (3) learning about sales and distributions options, e.g., sales via my own website, sales via Amazon, payment via PayPal or credit cards, and so forth. This has been very interesting, although the range of printing and sales/distributions options can be pretty confusing. What seems most clear to me at this point is that national, particular continental, boundaries still very much get in the way of global sales and distribution. For instance, I do not have a fully satisfactory way to distribute books within Europe; at present I plan to sell them via my website and PayPal (in US dollars) and ship each book to Europe from the United States. I wish I knew about some on-line discussion group in Europe that serves the same mutual-education function that the Yahoo-based Self-Publishing discussion group does for small US publishers.

FG: I'm sure you have plenty of new ideas for future publications?

DW: I believe my publishing experimentation will become more valuable as I publish a *few* more books: I plan to reissue an oral history of my mother, who is of Germans-from-Russia heritage and grew up in North Dakota speaking an obscure German dialect on a farm originally without running water or electricity; and I plan to finish compiling and probably publish myself a technology history of the company known as Bolt Beranek and Newman (BBN) where many innovations in computer technology and applications were accomplished. Who knows what will happen after that – perhaps some compendium of TeX-related writings.

Frans Goddijn
frans (at) goddijn (dot) com

The isodoc class
for letters, invoices, and more

Abstract
The isodoc class can be used for the preparation of letters, invoices, and, in the future, similar documents. Documents are set up with options, thus making the class easily adaptable to user's wishes and extensible for other document types.

Keywords
letter, invoice, key/value, NEN 1026

Introduction

This class is intended to be used for the preparation of letters and invoices. Its starting point was Victor Eijkhout's NTG `brief` class[1], which implements the NEN 1026 standard. The `brief` class does not provide facilities for invoices and it is not easily extensible.

The goal for the `isodoc` class is to be extensible and easy to use by providing *key=value* configuration. Furthermore, texts that need to be placed on prescribed positions on the page (there are many such texts) are positioned by using the `textpos` package.[2] This provides a very rugged construction of the page.

The class itself contains many general definitions, but variable data, such as opening, closing, address and many more, have to be defined using *key=value* definitions, either in the document or in a style file. The latter is indicated for definitions that don't vary on a per document basis, such as your company name, address, email address and so on. Thus if you run a company and are the secretary of a club, you would have style files for each of them, plus one for your private letters or invoices.[3]

The general setup for a source producing one or more letters is (see figures 1 –3 , page 94–95, for examples):

```
\documentclass{isodoc}
\usepackage{<somestyle>}
\setupdocument{<generaloptions>}
\begin{document}
\letter[<addressee_specific_options>]{<letter_content>}
... more \letter calls ...
\end{document}
```

Similarly, the general setup of a document producing one or more invoices is (figure 4 , page 98):

```
\documentclass{isodoc}
\usepackage{<somestyle>}
\setupdocument{<generaloptions>}
\begin{document}
\invoice[<addressee_specific_options>]{<invoice_content>}
```

```
... more \invoice calls ...
\end{document}
```

Options

As shown in the two examples in the previous section, there are three commands that can set options: `\setupdocument`, `\letter`, and `\invoice`. These commands will be further explained in the *Commands* section. `\setupdocument` is normally used to set options that are common to all letters of invoices in the document, like your company data; `\letter`, and `\invoice` set only those options that are different for each letter or invoice, like the `to` and `opening` options.

This section lists and explains all available options. All options can be used in both the style files and in the document source, although several will normally only be used in style files (such as `company`) and some only in the document source (such as `to` or `opening`).

Language
Currently five languages are defined. As I am not particularly strong in the translation of administrative terminology, please feel free to send me corrections. And if you don't find your own language here, please send me your translations and your language will be added. The keywords below set the language, English is used by default.

`dutch`	Set language to Dutch,
`english`	English,
`german`	German,
`american`	American,
`french`	or French.
`foreign`	Use this key if you send your letter to a foreign country. With it, your country will be added to return and logo addresses, your zip code will be prefixed with your country code, telephone numbers will be prefixed with +31– (or whatever your `areacode` option has been set to) instead of just a 0. In the `\accountdata` command, it causes IBAN en BIC code to be included.

Logo
Information about the sender is defined here. The logo, by default, consists of a large company name on top a rule, with a contact person's name (probably your own name) and address hanging under the rule.

`company = ...`	Your company name as it should appear in the logo (if you use the default logo) and in the return address (where it may get overridden by the `returnaddress` keyword.) For private documents, use your name or nickname here.
`who = ...`	Contact person's name; probably your own name.
`street = ...`	Street in the sender's address.
`city = ...`	City in the sender's address.
`zip = ...`	Zip in the sender's address.
`country = ...`	Country in the sender's address. Only used if `foreign` key was used.
`countrycode = ...`	Sender's country code. For The Netherlands: NL
`areacode = ...`	Sender's area code. For The Netherlands: 31

Address window
The addressee's address is printed in a window. The width of the window is two columns (70 mm), and its contents are vertically centered in it. There are no limits to the vertical size of the window, other than the physical size of the window in the envelopes you use. The vertical position of the window's center is set with the `addresscenter` keyword. Horizontally there are two options: left or right.

`leftaddress`	Places the window over columns 2 and 3; this is the default.
`rightaddress`	Places the window over columns 4 and 5.
`addresscenter = ...`	Distance in mm of the center of the window from the top of the paper; the default value is 63.5 mm, fitting for a DL envelope for triple folded A4 (110x220mm) with a window at 50 mm from the top, 30mm high.[4] Use `addresscenter` *after* `fold`, `fold2` or `fold3`, because the definition of the folding marks also puts the window positions at their corresponding defaults.
`to = ...`	The addressee's address. New lines can be introduced with the \\ command; lines longer than 70 mm will cause extra newlines.
`[no]return`	Do or don't print a return address on top of the addressee's address. This is useful if blank window envelopes are used. The return address is composed from the contents of the `company`, `street`, `zip`, `city`, and `country` keywords; it is printed in a bold script size sans serif font and is is separated from the addressee's address with a rule. The country will only be printed if the `foreign` keyword has been used.
`returnaddress = ...`	The return address, if it is composed as just described, may become too long to fit in the address window. Or you may want to define a completely different return address. With the `returnaddress` keyword you can redefine the return address. Use \\ to insert bullets.

Opening and Closing
A letter is started with an opening – something like 'Dear John', and ended with a closing – something like 'Regards,<newline>Betty', perhaps with an autograph (or white space) in between.

`opening = ...`	Dear John
`closing = ...`	Regards
`signature = ...`	Betty
`autograph = ...`	This keyword can have one of the 10 values 0–9: 0: no autograph; the `signature` appears right under the `closing`. 1: generates extra white space between `signature` and `closing` for a hand-written autograph. Change with the `closingskip` key. 2–9: inserts one of eight autograph images which, with the \autograph command, may have been defined in the style file.

`enclosures = ...` This keyword can be used to add a note, at the end of the document, which starts with **Enclosure:** followed by the value of the keyword. Multiple enclosures can be separated with \\ commands. If those are found, the starting text will be **Enclosures:**.

`closingskip = ...` white space between signature and closing, default: 2\baselineskip.

Headline fields

Under the address window, a headline is printed. The page is vertically divided in six columns, one each for the left and right margins, and four which, in the headline, say: *Your letter of*, *Your reference*, *Our reference*, and *Date*, each with their respective contents under them. If the `subject` keyword is used, an extra line starting with *Subject:* will appear, followed by the contents on the same line and over a width of 2.5 columns. If needed, extra lines will be used.

`yourletter = ...` first field in the headline: the date of the letter this document is reaction on; empty by default.

`yourref = ...` second field in the headline: addressee's reference of the letter this document is reaction on; empty by default.

`ourref = ...` third field in the headline: your own reference for this document.

`date = ...` fourth field of the headline; the default is 'Undefined date', i.e. the date of \today is not the default as this would make the date untraceable from the document source only. The argument of the `date` option must have the form `yyyymmdd`; it will be translated into a date like "May 3, 2006" if the document language is English, or into its translation in the actual language.

`subject = ...` subject of this document; is placed under the headline, over the width of the first, second and half of the third fields.

Footer fields

Footer fields are shown in the order in which they appear below; they are empty by default, and empty fields are not displayed.

`[no]footer` enables or disables printing a page footer; there is room for up to four fields, if you set five fields, the last one will appear in the right margin.

`phone = ...` if not empty, prints 'phone' in the first field of the footer, with the contents under it, prefixed with a 0 or, if the `foreign` option was used, the areacode (set with the `areacode` option.) Telephone numbers should thus be entered without a prefix.

`cellphone = ...` same for cellphone...

`fax = ...` fax...

`email = ...` email...

`website = ...` and website.

Folding marks
Folding marks can be useful, particularly if your address window is used to its limits. Correctly folding your letter then prevents parts of the address to become invisible because of the letter loosely filling the envelope.

`nofold`	Disable folding marks.
`fold2`	Folding mark at about halfway, set for tight fitting into a 220x162 mm envelope, with a tolerance of 2 mm at both sides.
`fold3`	Folding mark at about one third from the top, set for tight fitting into a 220x110 mm envelope, with a tolerance of 1.5 mm at both sides.
`fold = ...`	For non-standard envelopes and paper formats the position of the folding mark can be set at any position (in mm) from the top of the paper.

Payment data
In invoices you probably want to make clear where you want your debtor to transfer his money to. You can do so by calling the `\accountdata` command, which generates a little table containing these data. The contents of this table can be defined with the following keywords:

`accountno = ...`	Your bank account number.
`accountname = ...`	Your bank account's ascription; company name by default.
`iban = ...`	Your account's IBAN...
`bic = ...`	and BIC code; IBAN and BIC are only reported in invoices to foreign customers—see the `foreign` keyword.
`vatno = ...`	Your VAT reference number, not yet used.
`chamber = ...`	Your Chamber of Commerce subscription number, not yet used.

Accept data
These keys pertain to data needed for accept forms:

`acceptaccount = ...`	Payer's bank account number
`acceptaddress = ...`	Payer's address lines, separated with `\\`
`accepteuros = ...`	Euro part of the amount to be paid
`acceptcents = ...`	Cents part of the amount to be paid
`acceptdescription = ...`	Description to be quoted on the accept form
`acceptdesc = ...`	Short version of the description for the detachable strip of the form to be kept by the payer
`acceptreference = ...`	Reference

Miscellaneous
Normally, texts in letters are set ragged right. This can be changed with the following keywords:

`[no]fill`	Use the `fill` keyword to justify text both left and right; the default is `nofill`: left justification only.
`fontpackage = ...`	The default font is Latin Modern (`fontpackage = lmodern`), but with the `fontpackage` keyword you can select another package, like `txfonts` or `osf-txfonts`.

Commands

\showkeys

The \showkeys command can be useful for debugging. It prints a table showing the option keys described in the previous section, and their current values.

\setupdocument

Most of the setup, both in the style files and in the documents themselves, is done setting options in a call to the class-defined \setupdocument command. The options can be either a key/value pair, or just a key. Options with values and those without may occur in any order, with exception of the addresscenter option (see there.) Values need their surrounding {}'s only if they contain any comma's. The *Options* section explains the available options.

Most of the options have a corresponding command with the same name. Although not very often, it may sometimes be useful to have those commands available. These are the options with a corresponding command:

acceptaccount	areacode	email	returnaddress	yourref
acceptaddress	bic	enclosures	signature	zip
acceptcents	cellphone	fax	street	
acceptdesc	chamber	fontpackage	subject	
acceptdescription	city	iban	vatno	
accepteuros	closing	opening	website	
acceptreference	company	openingcomma	who	
accountname	country	ourref	addresscenter	
accountno	countrycode	phone	yourletter	

So you could write in your letter: "please send me the money on my bank account: \accountno\ as soon as possible."

\letter

The \letter command produces one letter and can be called multiple times. It has two arguments. The first argument is optional and must be a list of *key=value* pairs. The options set here are usually those that vary among different letters. The second argument contains the letter's content. This content will, depending on the options set, automatically be surrounded by an opening, a closing, an autograph, a signature and a remark about any enclosures. The first page of each letter will be decorated with a logo, the addressee's address, a return address, various reference fields, a footer, a folding mark—all as defined by *key=value* pairs in \setupdocument or in the \letter command itself.

The second an following pages will have a heading, quoting the name of the addressee and a page number. Examples of letters can be found in the section *Usage: letters*.

\invoice

The \invoice command is essentially the same as the \letter command, except that the opening is always "INVOICE", and the content (argument 2) is largely composed using the \itable, \iitem, \itotal, and \accountdata commands described hereafter. Closing, autograph, and signature are disabled.

In the Netherlands, invoices can be provided with an accept form on the lower third part of the page. If the accept option was used, this accept form will be filled with the available data, in the ocrb font where needed.

The following commands pertain to invoices:

\itable

The \itable command uses tabularx to create a two-column table. The first column of the table will have the header 'Description' (or its equivalent in the language selected), the header of the second column says 'Amount (€)'. The single argument of \itable should contain the contents of the table and is of the form:

```
item 1 & amount 1\NN
item 2 & amount 2\NN
...
item n & amount n \NN
\cmidrule[.05em]{2-2}
Total  & amount \NN
```

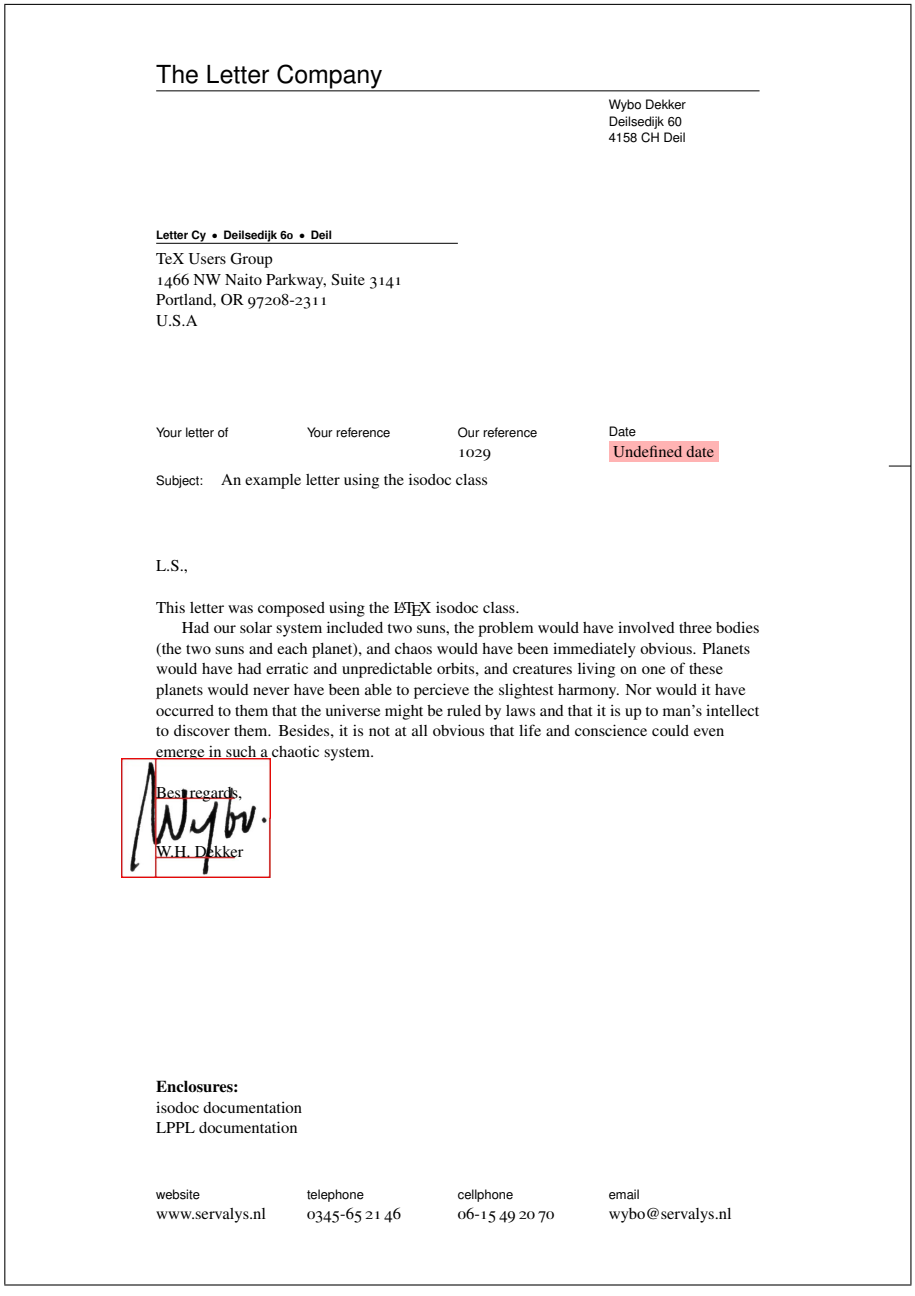


Figure 1. Minimal letter example

However, the next two commands may be used to enter these data more cleanly:

\iitem The `\iitem{item}{amount}` command (`iitem` stands for Invoice Item) is easier way to write "`item & amount\NN`".

\itotal The `\itotal{amount}` command (`itotal` stands for Invoice total) is equivalent to writing "`\cmidrule[.05em]{2-2} Total & amount \NN`", with the additional advantage that the word 'Total' will be replaced with its equivalent in the current language. Thus, the argument to the `\itable` command show above can also be written:

```
\iitem{item 1}{amount 1}
\iitem{item 2}{amount 2}
...
\iitem{item n}{amount n}
\total{amount}
```

\accountdata The `\accountdata` command prints a little table with accounting information the creditor needs for paying the invoice. It is constructed using the values of the options `accountnumber`, `accountname`, `iban`, and `bic`. The latter two are only included if the `foreign` option was used.

\autograph The `\autograph` command, which will normally appear in a style file, serves to define up to eight autographs based on PDF, JPEG or PNG images. One of these autographs will be drawn between the closing (*Best regards*) and the signature (*Betty*) if you use the `autograph` option with a value from 2 through 9. `\autograph` has 7 arguments:

arg 1: 2,3,...9: autograph number; will be translated internally to define `\autographA`, `\autographB`... `\autographH`
2: scaling factor for the image
3: distance the autograph outdents in the margin
4: vertical position of the baseline of the closing (*Regards,*) from the top
5: vertical position of the baseline of the signature (*John Letterwriter*) from the top
6: height of the image
7: the image (jpg, png, pdf...)

The arguments 3–6 must be dimensions, and for a given autograph image should be inferred by inspecting the image with an image manipulation program like, for example, the gimp. In the lower left corner of the gimp window, select the units of length, move the pointer to the positions where you want margin, closing, and signature and to the bottom of the image, read the x, y, y and y positions respectively and use those for the argument 3, 4, 5, and 6.

\logo The `\logo` command is internally used to define the default logo; you can redefine it with `\renewcommand{\logo}{...}`. An example of logo redefinition can be found on page 97.

Usage: letters

Usage of the class is best explained with an example. Here is the latex source for a small letter; its result appears in figure 1 :

```
\documentclass[11pt]{isodoc}
\usepackage{mystyle}
\setupdocument{
       to  = {TeX Users Group\\
              1466 NW Naito Parkway, Suite 3141\\
              Portland, OR 97208-2311\\
```

```
              U.S.A
             },
    ourref  = 1029,
 enclosures = isodoc documentation\\LPPL documentation,
   subject  = An example letter using the isodoc class,
 autograph  = 2,
}

\begin{document}
\letter{This letter was composed using the \LaTeX{} isodoc class.
  \par\input{thuan}
}
\end{document}
```

This source essentially shows three items:

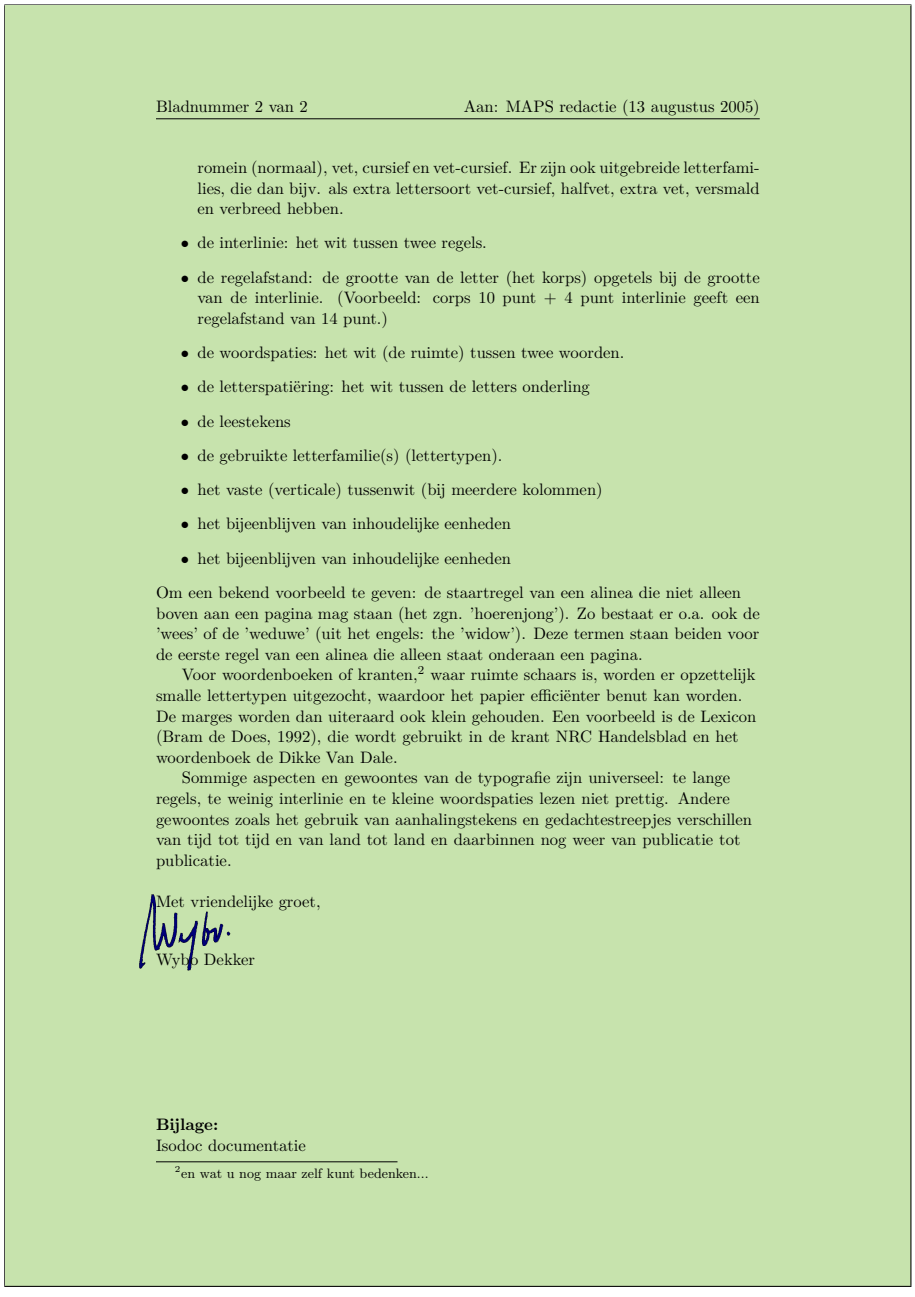
- the inclusion of a package `mystyle`; we'll come to that shortly.
- the command `\setupdocument` called with many *key=value* arguments, each defining one of the texts that go into the letter.
- the command `\letter`, enclosing the body of the letter; just to give the letter some real body, a small text has been included using `\input`.

Of course this is not all of the information needed to create a letter. For example, there should be a logo, telling the addressee who I am and there should be contact information such as my address, telephone number and so on. This is where the included `mystyle` package plays its part. Here is an example of such a style file:

```
\NeedsTeXFormat{LaTeX2e}[1999/12/01]
\ProvidesPackage{mystyle}
  [2006/04/04 v1.0 Letter Company style file for isodoc]

\setupdocument{return,footer,fold3,
  fontpackage     = osf-txfonts,
  autograph       = 0,
  company         = The Letter Company,
  returnaddress   = Letter Cy\\Deilsedijk 60\\Deil,
  who             = Wybo Dekker,
  street          = Deilsedijk 60,
  city            = Deil,
  zip             = 4158 CH,
  country         = The Netherlands,
  countrycode     = NL,
  areacode        = 31,
  phone           = {345-65\,21\,46},
  cellphone       = {6-15\,49\,20\,70},
  fax             = {},
  website         = www.servalys.nl,
  email           = wybo@servalys.nl,
  accountno       = {3040\,46221},
  iban            = nl61pstb0006238747,
  bic             = pstbnl21,
  vatno           = 28750482B01,
  chamber         = 11023220,
  opening         = L.S.,
  closing         = Best regards,
  signature       = W.H.~Dekker
}
\graphicspath{{./graphics/}}
\autograph{2}{.30}{75bp}{87bp}{216bp}{261bp}{signw_marked}
\endinput
```

Figure 2. Long letter example with a non-standard logo, page 1



Bladnummer 2 van 2 Aan: MAPS redactie (13 augustus 2005)

romein (normaal), vet, cursief en vet-cursief. Er zijn ook uitgebreide letterfamilies, die dan bijv. als extra lettersoort vet-cursief, halfvet, extra vet, versmald en verbreed hebben.

- de interlinie: het wit tussen twee regels.
- de regelafstand: de grootte van de letter (het korps) opgetels bij de grootte van de interlinie. (Voorbeeld: corps 10 punt + 4 punt interlinie geeft een regelafstand van 14 punt.)
- de woordspaties: het wit (de ruimte) tussen twee woorden.
- de letterspatiëring: het wit tussen de letters onderling
- de leestekens
- de gebruikte letterfamilie(s) (lettertypen).
- het vaste (verticale) tussenwit (bij meerdere kolommen)
- het bijeenblijven van inhoudelijke eenheden
- het bijeenblijven van inhoudelijke eenheden

Om een bekend voorbeeld te geven: de staartregel van een alinea die niet alleen boven aan een pagina mag staan (het zgn. 'hoerenjong'). Zo bestaat er o.a. ook de 'wees' of de 'weduwe' (uit het engels: the 'widow'). Deze termen staan beiden voor de eerste regel van een alinea die alleen staat onderaan een pagina.

Voor woordenboeken of kranten,[2] waar ruimte schaars is, worden er opzettelijk smalle lettertypen uitgezocht, waardoor het papier efficiënter benut kan worden. De marges worden dan uiteraard ook klein gehouden. Een voorbeeld is de Lexicon (Bram de Does, 1992), die wordt gebruikt in de krant NRC Handelsblad en het woordenboek de Dikke Van Dale.

Sommige aspecten en gewoontes van de typografie zijn universeel: te lange regels, te weinig interlinie en te kleine woordspaties lezen niet prettig. Andere gewoontes zoals het gebruik van aanhalingstekens en gedachtestreepjes verschillen van tijd tot tijd en van land tot land en daarbinnen nog weer van publicatie tot publicatie.

Met vriendelijke groet,

Wybo Dekker

Bijlage:
Isodoc documentatie

[2] en wat u nog maar zelf kunt bedenken...

Figure 3. Long letter example with a non-standard logo, page 2

So in the style file, too, \setupdocument is used to register information that will common to almost all of my letters. The \autograph command sets up an autograph, based on an image file. Apart from the code shown here, a style file can contain definitions for more autographs, and a definition for a logo. Without the latter, a default logo is produced. Note also that I have included defaults for opening, closing, and signature in the style file, and that I did not override those in the letter's source.

The letter source example shown above, in combination with this style example, compiles to the letter shown in figure 1 . This example illustrates some aspects of isodoc:

- At the top, you see the default letterhead (logo). Create your own logo with \renewcommand{\logo}{...}.
- Under it is the address. It has a return address in script sized sans serif boldface over it, because the return key has been used. A return address is useful if you send your letters in a standard window envelope. The positioning of the address is done in the style file, using the addresscenter and leftaddress or rightaddress keywords.
- The paper is vertically divided in six equally wide columns. The outer two columns are the left and right margins, the second to fifth columns contain header and footer fields.
- The "Your reference" and "Our reference" fields have not been set (with the yourref and ourref keys) and therefore stay empty by default, the date field has also not been set, but it should be. Therefore, the default value is "Undefined date", and a warning is issued by a pink background.
- A folding mark has been printed in the extreme right margin, such that on folding the paper along it, it will correctly fit in a 220 x 110 mm envelope; this has been achieved by using the fold3 key.
- In between closing (*Best regards,*) and signature (*W.H. Dekker*) an autograph has been placed. This was done by setting autograph=2. Alternative values are 0 (nothing between closing and signature), 1 for white space where an autograph can be placed with a pen after printing, or one of the values 2–9, which may have been associated with other autograph images. In this case, I have used an autograph image in which I have drawn the boundary box and the *outdent*, *closing*, and *signature* positions defined in the \autograph command (see the section *Commands*) with red lines.
- The bottom of the letter has (up to) four fields with contact information. This is useful if your logo does not show that information. If it does, you can omit these fields by using the nofooter key, or by not using the footer key, depending on the default set in the style file.

Let's try another illustrative example, see figures 2 and 3 (page 94): we use a modified style file, with a redefined logo, so we don't need a page footer; we use preprinted right-windowed envelopes, so a return address is not needed. Here is the style file (logostyle.sty):

```
\setupdocument{
  nofooter,fold2,autograph=1,dutch,
  company         = The Shiva Shakti Foundation,
  who             = Wybo Dekker,
  street          = Deilsedijk 60,
  city            = Deil,
  zip             = 4158 CH,
  country         = The Netherlands,
  countrycode     = IN,
  areacode        = 31,
  phone           = {345-65\,21\,46},
  cellphone       = {6-15\,49\,20\,70},
```

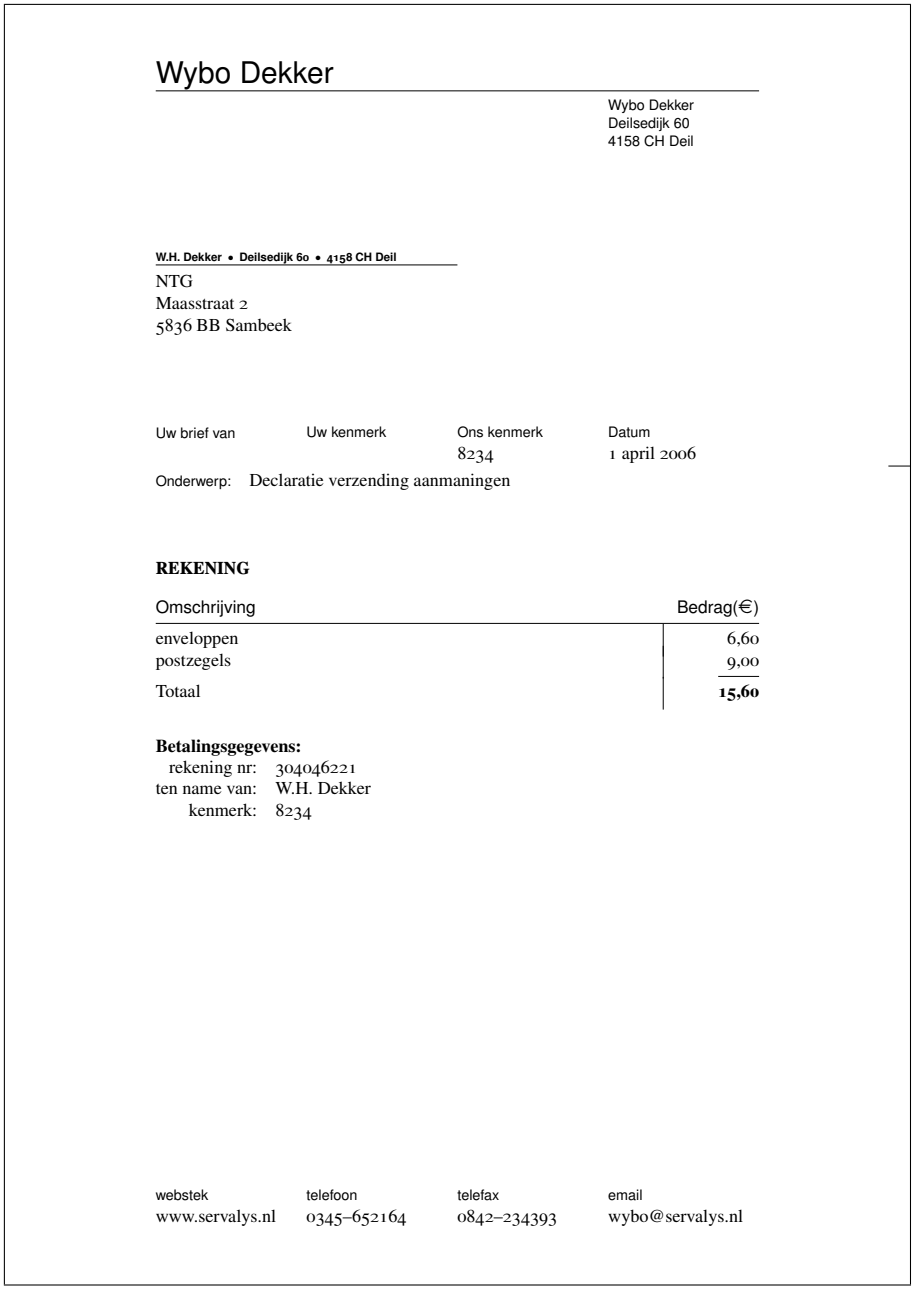
```
  fax             = {},
  website         = www.servalys.nl,
  email           = wybo@servalys.nl,
  accountno       = {3040\,46221},
  iban            = nl61pstb0006238747,
  bic             = pstbnl21,
  addresscenter   = 70,
  rightaddress
}
\autograph{2}{.20}{75bp}{47bp}{238bp}{261bp}{signblue}

\definecolor{shivablue}{rgb}{.14,.33,.43}
\definecolor{shivaback}{rgb}{.78,.89,.68}
\pdfmapfile{=chopinsc.map}
\newcommand{\chopinscript}{\fontfamily{chopinscript}\selectfont}
\DeclareFontFamily{T1}{chopinscript}{}
\DeclareFontShape{T1}{chopinscript}{m}{n}{ <-> chopinsc}{}
\graphicspath{{./graphics/}}

\renewcommand{\logo}{
  \pagecolor{shivaback}
  \begin{textblock}{105}(88,15)
    \begin{center}
      \chopinscript{%
        \Huge\noindent
        \textcolor{shivablue}{The Shiva Shakti Foundation}
      }\\[2ex]
      Main Building\quad 567\textsuperscript{th}
      floor\quad Room 123\quad Bangkok
    \end{center}
  \end{textblock}
  \begin{textblock}{2}(10,13)
    \includegraphics[scale=.3]{shiva_shakti.jpg}
  \end{textblock}
}
\endinput
```

The letter source does not use the autograph key, so the default value of 2 is used; we write it in Dutch and use a larger text, just to see what happens if more than one page is generated:

```
\documentclass[11pt,twoside]{isodoc}
\usepackage{logostyle}
\setupdocument{
    ourref = 1029,
yourletter = 12 mei,
   yourref = MAPS \#34,
      date = 20050813,
   closing = Met vriendelijke groet,
 signature = Wybo Dekker,
enclosures = Isodoc documentatie,
   subject = Voorbeeldbrief met de isodoc class,
 autograph = 2,
}
\newcommand{\letterbody}{%
  Dit is een voorbeeld van een brief gemaakt met de isodoc class.
  Het plaatje in het logo werd ontworpen door Pieter Weltevrede.
  De tekst bestaat uit wat bijelkaar geraapte
  teksten\footnote{opgevist uit een van de voorbeeld-teksten uit
  de \TeX-distributie} om een lange brief te krijgen.
```



Wybo Dekker

Wybo Dekker
Deilsedijk 60
4158 CH Deil

W.H. Dekker • Deilsedijk 60 • 4158 CH Deil

NTG
Maasstraat 2
5836 BB Sambeek

Uw brief van	Uw kenmerk	Ons kenmerk	Datum
		8234	1 april 2006

Onderwerp: Declaratie verzending aanmaningen

REKENING

Omschrijving	Bedrag(€)
enveloppen	6,60
postzegels	9,00
Totaal	**15,60**

Betalingsgegevens:
rekening nr: 304046221
ten name van: W.H. Dekker
kenmerk: 8234

webstek	telefoon	telefax	email
www.servalys.nl	0345–652164	0842–234393	wybo@servalys.nl

Figure 4. Invoice example

```
    \par\input{typo}
  }

  \begin{document}
  \letter[to = Wybo Dekker\\
               Deilsedijk 60\\
               4158 CH Deil,
               opening = Beste Wybo
         ]{\letterbody}
  \letter[to = MAPS redactie\\
               Spuiboulevard 269\\
               3311 GP Dordrecht,
               opening = Beste Taco
         ]{\letterbody}
  \end{document}
```

In this case, the same letter had to be sent to two different people, with different openings and addresses of course. So the letter's body is separately defined and the `\letter` command is called twice, with the same body, but different `to` and `opening` keys. Figures 2 and 3 show the first two pages (the first letter) of this document, which actually has four pages.

Usage: invoices

Invoices (can) have the same structure as letters, except that the `\opening` isn't "Dear Somebody" anymore, but something like "Invoice". And the `\closing` does not say "Best regards", but may provide payment information. And the body is not a simple text, but a table with descriptions of things to be paid, and the corresponding amounts of money.

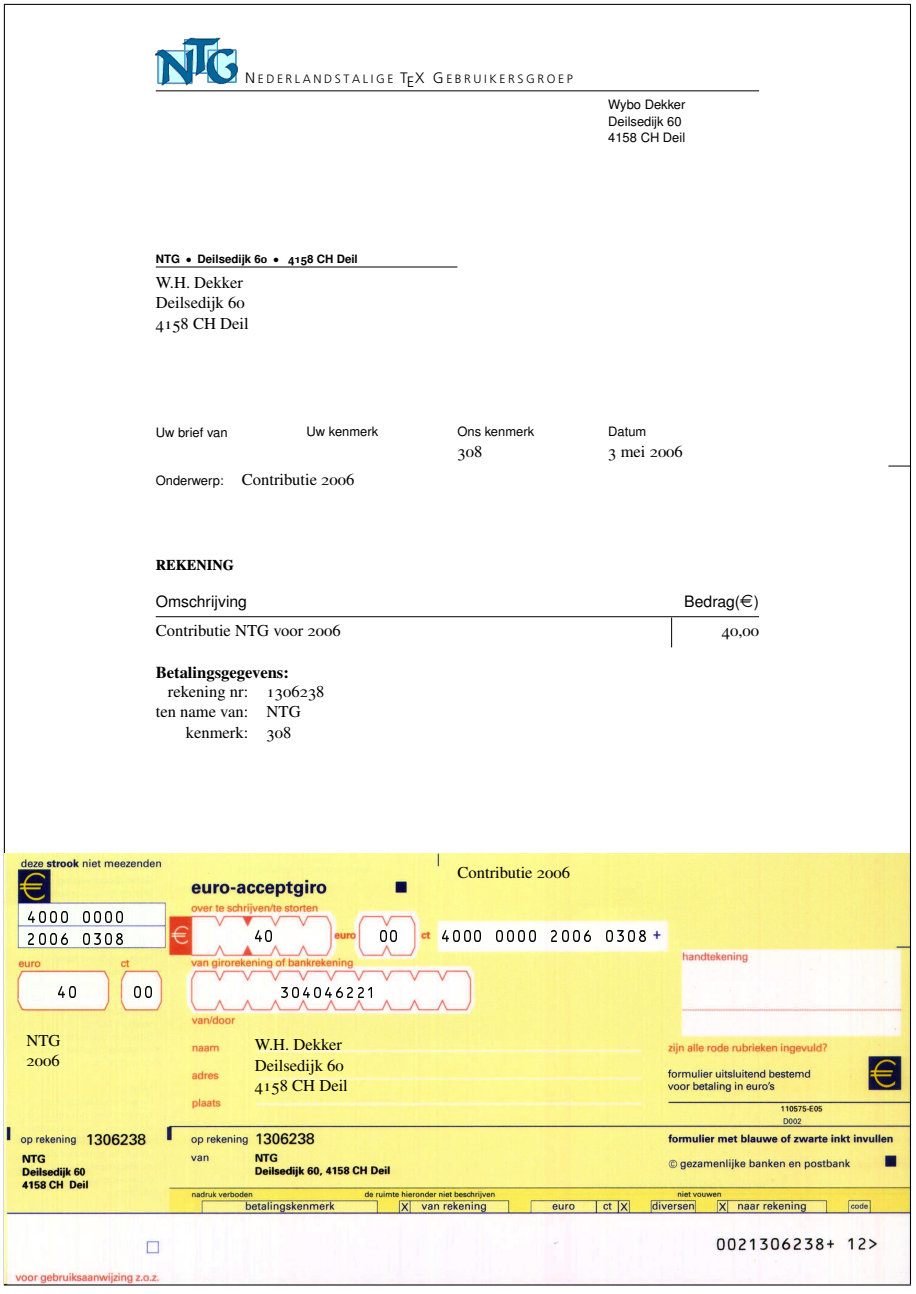
An example, as usual, is most instructive:

```
\documentclass[12pt]{isodoc}
\usepackage{isowybo}
\setupdocument{
  ourref=8234,
  date=20060401,
  subject=Declaratie verzending aanmaningen,
  to=NTG\\Maasstraat 2\\5836 BB Sambeek
}
\begin{document}
\invoice{%
  \itable{
    \iitem{enveloppen}{6,60}
    \iitem{postzegels}{9,00}
    \itotal{15,60}
  }
  \\[3ex]\accountdata
}
\end{document}
```

The result is shown in figure 4 .

When the `accept` option is used, the invoice will be created with an invoice form on the lower third part of the page. Here is an example:

```
\documentclass[11pt]{isodoc}
\usepackage{isontg}
\setupdocument{accept,
  acceptdesc=NTG\\2006,
  acceptdescription=Contributie 2006,
```



NEDERLANDSTALIGE TEX GEBRUIKERSGROEP

Wybo Dekker
Deilsedijk 60
4158 CH Deil

NTG • Deilsedijk 60 • 4158 CH Deil

W.H. Dekker
Deilsedijk 60
4158 CH Deil

Uw brief van	Uw kenmerk	Ons kenmerk	Datum
		308	3 mei 2006

Onderwerp: Contributie 2006

REKENING

Omschrijving	Bedrag(€)
Contributie NTG voor 2006	40,00

Betalingsgegevens:
rekening nr: 1306238
ten name van: NTG
kenmerk: 308

deze **strook** niet meezenden

euro-acceptgiro

Contributie 2006

4000 0000
2006 0308

over te schrijven/te storten

40 euro 00 ct 4000 0000 2006 0308 +

van girorekening of bankrekening

304046221

handtekening

euro 40 ct 00

van/door

NTG
2006

naam W.H. Dekker
adres Deilsedijk 60
plaats 4158 CH Deil

zijn alle rode rubrieken ingevuld?

formulier uitsluitend bestemd voor betaling in euro's

110575-E05
D002

op rekening 1306238
NTG
Deilsedijk 60
4158 CH Deil

op rekening 1306238
van NTG
Deilsedijk 60, 4158 CH Deil

formulier met blauwe of zwarte inkt invullen

© gezamenlijke banken en postbank

nadruk verboden — de ruimte hieronder niet beschrijven — niet vouwen

betalingskenmerk | X | van rekening | euro | ct | X | diversen | X | naar rekening | code

0021306238+ 12>

voor gebruiksaanwijzing z.o.z.

Figure 5. Invoice example with accept form

```
    acceptreference=4000 0000 2006 0308,
    date=20060503,
    subject=Contributie 2006,
    nofooter
  }
  \begin{document}
  \invoice[
    to=W.H. Dekker\\Deilsedijk 60\\4158 CH Deil,
    acceptaccount=304046221,
    acceptеuros=40,
    acceptcents=00,
    ourref=308,
  ]{\itable{\iitem{Contributie NTG voor 2006}{40,00}}\\[3ex]
    \accountdata
    \begin{textblock}{210}(0,197)
      \noindent\includegraphics[width=210mm]{acceptform.jpg}
    \end{textblock}
  }
  \end{document}
```

Normally such invoices are printed on preprinted paper with an easily detachable, perforated form. In this example, the form itself has been printed, too. The `graphicx` and `textpos` packages have already been made available by the `isodoc` class. Figure 5 shows the output of this example.

Notes

1. CTAN: ntgclass/briefdoc.pdf
2. CTAN: textpos/textpos.pdf
3. If you archive your documents in their source form only, it may be wise to work without a style file and set all options in the document itself!
4. The middle of the window is at 50+30/2=65 mm from the top of the envelope; the paper is folded (see the folding options below) to give the folded paper a tolerance of 1.5mm on both sides in the envelope, so the address should be placed 1.5 mm higher at 65-1.5=63.5 mm.

Wybo Dekker
wybo (at) servalys (dot) nl

Creating a Dust-cover in ConTEXt
a layout exercise

Abstract
This short article describes how to setup a dust-cover for a book, using the standard features available in ConTEXt.

Keywords
non-guru ConTEXt dust-cover

Introduction

A beautiful set of narrations, originally written by my grandfather (father of my mother), a former police officer and tracker dog guide, inspired me to typeset them using ConTEXt, and have them published in a real book[1].
It turned out to be quite a challenge, because I'm not an expert when it comes to using ConTEXt. Manuals are always close at hand! Quality is my main incentive to use TEX and its progeny ConTEXt.

Book Design

The proportion between the width (140mm) and height of the page is set to 2:3. The typesetting area was obtained using the Tschichold[1] method, and equals 93.3mm by 140mm. The header contains the title of the book on the leftside of the left-hand page, while the title of the current chapter is being displayed at the rightside of the righthand page. The footer holds the pagenumber just in the middle. Each left-hand page that starts a new chapter has no header. An example page of the book is presented in Figure 1.

The book's contents, preface and colofon are typeset in Computer Modern Roman (12pt). The header and footer use the italic variant at 10pt.

Dust-cover

Computer Modern Roman (12pt) is also the main font style used in typesetting the dust-cover. The flaps however apply the sans serif style (10pt).

The dust-cover actually consists of two pages that are combined using the page arrangement feature of ConTEXt:

```
\setuparranging[2*2,doublesided]
```


Politie, politie!

HET MEEST VERRASSENDE uit de periode 'Buggenum' zou ik echter vergeten te vertellen:

Alzo was het in de ochtend van mijn vertrek uit Buggenum (10 juni 1924) dat een drietal arbeiders met twee vier-wielige wagens bespannen met paarden zich aan onze woning vervoegden. Tot mijn grote verbazing deelden deze mensen mij mee dat zij in opdracht van de voorzitter van Excelsior (M. Boonen) mij kosteloos moesten verhuizen en installeren. Ik gaf te kennen reeds met transportondernemer Hoefnagels uit Roermond te hebben afgesproken omtrent mijn verhuizing. Hierop werd mij echter meegedeeld dat deze afspraak door de voorzitter ongedaan was gemaakt en dat alzo Hoefnagels niet kwam. Mijn

101

Figure 1. Example page.

The above command puts the pages one and four next to each other. The page layout is shown in Figure 2.

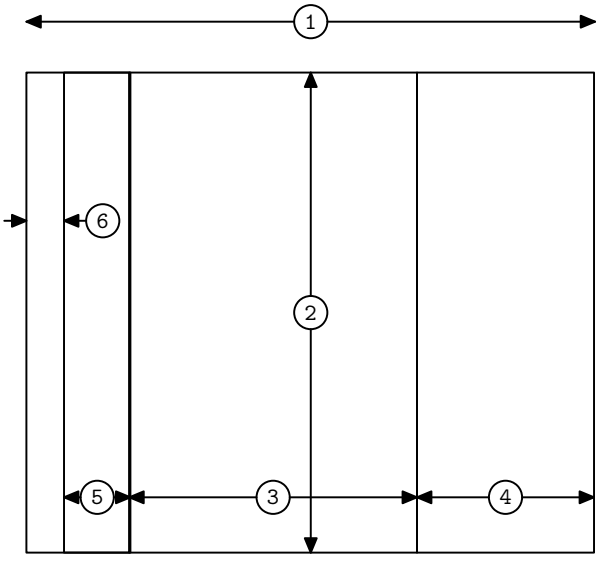
The dust-cover has two flaps and is based on a papersize of 257mm by 217mm, and defined by the following command:

```
\definepapersize[DUSTCOVER]
  [width=257mm, height=217mm]
```

I used the front flap for a short description of the book's intention, while the back flap contains a biography of the author in a nutshell! At the top of the backside one finds a photograph of the author together with a few key features of the book. The bottom shows the publisher's logo and the ISBN-number. The front of the cover shows a typical bridge in Griendtsveen, a municipality in the Peel. The Peel is the name of a marshy region in the west of the province Limburg. My grandfather has worked in this region for 10 years.

Grandparents' children cherish happy feelings when they recall their childhood in the Peel! Nowadays it is a region rich in natural beauty and worth exploring on foot or by bicycle.

The height of the linen hardcover exceeds that of the book's page by 3.5mm on both ends. The cover's width is 7mm larger than the span of the page. The size of the book's spine, 34mm, was obtained from the very first bounded copy. This of course equals the actual thickness of the book. The dust-cover should fit the linen hardcover as accurately as possible, so its dimensions equal those of the linen hardcover.



1: paperwidth = 257mm 2: paperheight = 217mm
3: textwidth = 130mm 4: rightmargin = 80mm
5: leftmargin = 30mm 6: leftedge = 17mm

Figure 2. The page layout for the dust-cover.

Front
I used the layer mechanism to create the front. It enabled me to position the text on top of the full-page photograph. The photograph has been adapted using the GIMP package, my favorite when it comes to producing and modifying images.

```
\definelayer[frontpicture][position=no]
\setlayer[frontpicture][x=-0.0mm,y=0.0cm,
                        preset=righttop]{%
 \externalfigure[bruggriendtsveen.jpg]
  [scale=1000]
}

\setupbackgrounds[text]
 [background=frontpicture]
```

The above given commands define and position the photograph on the front side of the cover.

The title is divided in a horizontal and vertical part. The `\startnarrower` command is applied to facilitate in a reduced width of the typesetting area.

Here is how it is realized:

```
\startnarrower[2*left,3*right]
 \rightaligned{\Large \color[MyColor]
  {Wat is    }\kern.5cm}
 \rightaligned{\rotate[rotation=270]
  {\Large \color[MyColor]
  {het leven kort!}\kern.5cm}}
```

The fontsize `Large` and color `MyColor` are defined in the preamble with `\definefont` and `\definecolor` respectively.

Spine
The code fragment displayed below is responsible for the text on the spine of the cover. It is used in combination with a layer.

```
\setupbackgrounds[header,text,footer][leftedge]
 [background=back]

\definelayer[back][
 width=2\edgewidth,
 height=\paperheight]

\setlayer[back][x=-17mm,y=0mm]
 {\rotate[rotation=270]
  {\framed[
   width=\paperheight,
   height=34mm,
   background=color,
   backgroundcolor=MyColor,
   foregroundcolor=white]
   {\bfd Wat is het leven kort! -
    P.G.Th. Sieben}}
 } % End of layer
```

Usage of a layer in this situation enabled the exact alignment of the text at the middle of the spine's width! Note the shift in the x-direction by -17mm, causing the layer to pass halfway across the edge of the page!

Back
The back also uses a reduced width of the typesetting area. The text is wrapped around a figure using the `left` -option.

The final result of the full-color cover is presented in Figure 3.

Flaps
The front flap is typeset using the `\inright` command with the settings defined as

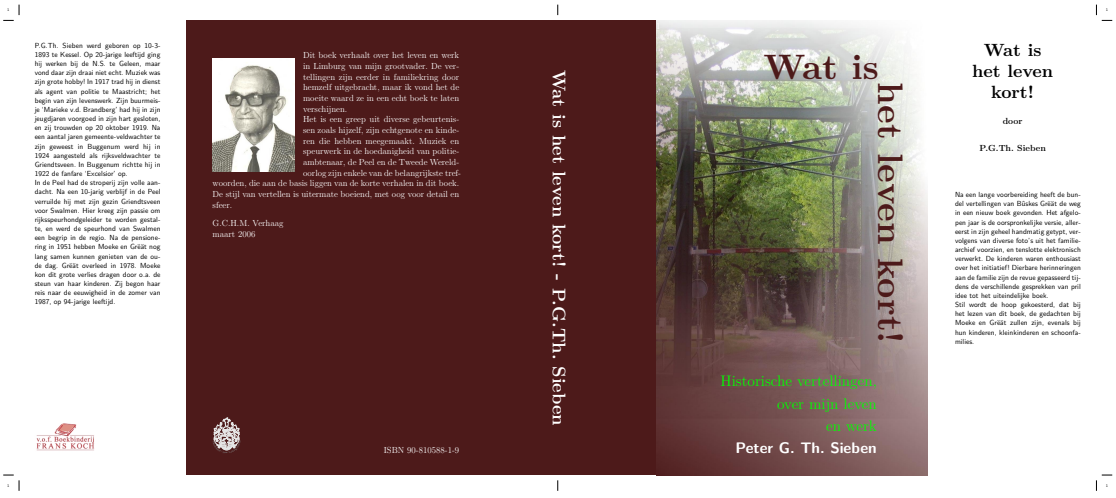


Figure 3. The final look of the cover.

```
\setupinmargin[right]
  [style=\ss\tfx\setupinterlinespace,
   align=no,width=60mm,distance=12.5mm]
```

The flap at the back is based on the `\inleft` command.

Conclusion

Creating a dust-cover using ConTEXt turned out to be fairly easy. It took me quite a while however, before I ended up with the solution as described here.

References

[1] Egger, W., *Help! The Typesetting Area*. Maps 2004, **30**, pp. 52-9

1. For those interested in the bookbinder, here is the information:
v.o.f. Boekbinderij Frans Koch, De Koningstraat 66, 5984 NJ Koningslust (`www.boekbinderijkoch.nl`)

Acknowledgement

I wish to thank the TEX world for providing us all with just great typesetting tools. The worldwide altruistic dedication of so many enthousiastic people to the numerous TEX based tools is just great!

Geert C.H.M. Verhaag
St. Jorisstraat 29
5954 AN Beesel
Netherlands
`verhaaggchm (at) ision (dot) nl`

TUG 2006 Report

Abstract
After TUG 2003 in America (Hawaii – USA), TUG 2004 in Europe (Xanthi – Greece), TUG 2005 in Asia (Wuhan – China), the TUG 2006 was held in Africa, more precisely in Marrakesh. Processing multilingual e-documents went beyond the limits of its traditional cultural areas and new horizons in the internationalization of TEX were explored.

Keywords
TUG 2006, Marrakesh, Arabic, international user meeting

Introduction

The TEX User Group's 27th annual meeting and conference was organized in collaboration with the Computer Science Department of the Faculty of Sciences Semlalia, Cadi Ayyad University, located in Marrakesh, Morocco. About thirty talks were held on three days, starting on November 9th, 2006. The conference was attended by more than forty people from all over the world.

The main topic and subtitle of the conference was: 'Digital Typography & Electronic Publishing: Localization & Internationalization'. It follows that there was a large focus on Arabic typography, but even so, a number of interesting presentations on altogether different subjects were given. A short step by step overview follows.

Wednesday November 8, arrival

Hans Hagen and I arrived (together with a few other people) at the airport of Marrakesh late in the morning, where we were picked up and chauffeur–driven to our hotel. Luckily, the hotel was only a few hundred meters away from the conference location, but that did not stop us from taking more than an hour to walk over there.

There was officially no program for this day, but because any of the international attendants arrived in the morning and early afternoon, the local organization had thoughtfully scheduled a tour of Marrakesh city by (open) doubledecker bus. As luck would have it, during this tour we had the only rainfall of the entire week, so we all got wet and had to hurry inside the bus. But from then on, we had a steady plus 20 degrees celcius and lots of sunshine, so no complaints about the weather.

Thursday November 9, day 1

The day started early, with the official welcoming speeches starting at half past eight. We were welcomed by UCAM (the university itself) as well as FSSM (the hosting faculty), and of course by TUG.

The first talk of the day was by Thomas Feuerstack and Klaus Höppner, who presented "ProTEXt, a complete TEX system for beginners". Most Windows-using readers will have seen the ProTEXt disk in last year's TEX Collection. The very easy to install and use TEX system based on MikTEX and TEXnicCenter will of course also be included this year. They are still looking for a Dutch translator for the installation manual, so if you want to help them out, please send me a message.

Next was my presentation of the new developments for the upcoming MetaPost release. The most important news is that MetaPost can now do font re-encoding and subset-ting. You can read all about that elsewhere in this (Maps #34) issue.

Jean-Michel Hufflen talked about the use of person names in his program MlBibTEX versus the traditional BibTEX. MlBibTEX (Ml stands for Multilingual) features

much improved input and handling of people's names. His quite colorful slides demonstrated a number of problematic names along with their usage in MlBibTEX.

The other two Dutch people present were Renée van Roode and Gerben Wierda, and together they talked about 'TEX Live – Life with TEX'. Renée produced a very nice overview of the past six years that Gerben has been working on the i-Installer and i-Packages for gwTEX. After that, Gerben took over with a more technical explanation of what the i-Installer does. It came as bit of a shock to most of us that the last slide was titled 'I Quit'. You can read more about his reasons why and what will happen to i-Installer elsewhere in the Maps.

Moroccan lunches are not be taken lightly. Or perhaps I should say: Moroccan lunches cannot be taken in lightly. The food has been excellent all through the conference, and especially so the lunches. These were served in a restaurant that was situated behind an unadorned garden door in a residential street. Considering the outstanding quality (and quantity) of the food served there, it is no surprise that they keep it a well guarded secret!

The afternoon program started with Hans Hagen telling everybody about the Mathadore project. The Dutch Mathadore project uses OpenMath to provide courseware for secondary education, and this content is typeset by ConTEXt after and intermediate conversion to MathML using an XSLT script that is part of the ConTEXt distribution.

Next up, Jerzy Ludwichowski talked about how Copernicus University in Toruń uses TEX to handle the admission forms for students. These legal documents are generated automatically from the base data in the administrative system, handling tens of thousands of admissions each year, with a minimum of effort.

Last talk of the day was Zdeněk Wagner: Babel speaks Hindi. The presentation described how the Babel module for Hindi was made, and it showed some of the problems associated with typesetting Hindi. For instance, in this script letters are reordered in the output, creating the need for a preprocessor to facilitate the input.

In the evening, there followed another visit to the old city of Marrakesh. We did a walk through the Souks, and returned to the hotel by means of horse and carriage.

Friday November 10, day 2

The Friday was completely focused on Arabic typography. The first talk was Yannis Haralambous. His talk, 'Infrastructure for High–Quality Arabic Typesetting' kicked off the day by unveiling the plans for Arabic typesetting support in the coming Omega 2. He showed us the planned data-structure and the algorithmic steps that will be used to produce high quality output in a large number of different languages that use the Arabic script.

Youssef Jabri talked about 'The arabi system – TEX writes in Arabic and Farsi'. Arabi is a newly arrived package that provides Arabic script support for LaTEX without the need for an external preprocessor or a special TEX extension. It adds support for the Arabic and Farsi languages to babel, it comes with a set of suitable fonts, and it understands a number of different input encodings. Support for Tifinagh, Syriac and even Urdu is planned for the near future.

Karel Píška demonstrated some interesting techniques that can be used to improve existing font technologies. The focus of 'Outline font extensions for Arabic typesetting' was on using a dynamic representation of the glyph shape so as to allow run–time generation of curvilinear keshideh connections instead of the rule-based fillers that are common today.

After the break, Hossam Fahmy introduced us to his system that aims at typesetting the Qur'an: AlQalam ("the pen" in Arabic). The system evolved out of modifications to ArabTEX. Much work is still needed, especially in the area of fonts, but also when it comes to low–level support from the typesetting program. Nevertheless, the intermediate results are already quite impressive.

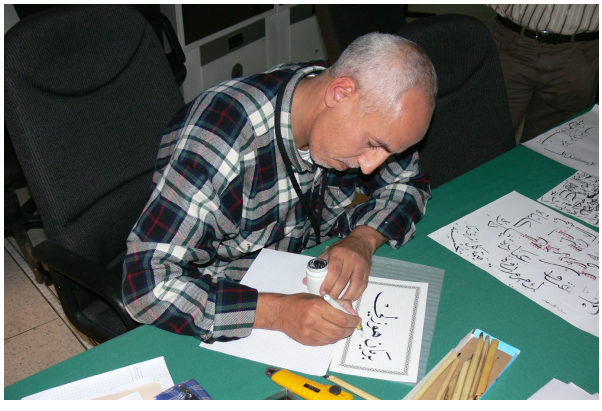
Have you ever felt the need to key in LaTeX commands in your native language? Well, if you are speaking Arabic, now you can! Mustapha Eddahibi presented 'DadTeX – A full Arabic interface'. This project makes it much easier to use LaTeX for Arabic-speaking people that are not familiar with the English language.

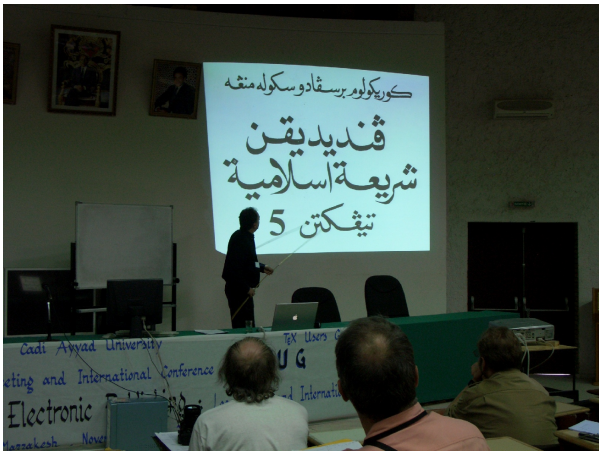
Next in line was Taco Hoekwater who acted as stand-in for Idris Hamid. He summarized the Oriental TeX project, its objectives and time-line. He also explained the relationship between this project that is sponsored by the University of Colorado State (USA) and the pdfTeX successor LuaTeX as well as upcoming MetaPost versions.

After yet another impressive lunch, Jonathan Kew demonstrated some of the multi–lingual capabilities of XeTeX. Besides a set of 'prepared earlier' slides, he also gave an impressive live demonstration. The second part of his talk was all about the growing Unicode math support in XeTeX. This will no longer need special TeX fonts but is able to use the new Unicode support found in the latest OpenType fonts.

Mohamed Elyaakoubi presented research on the justification of Arabic text. Hyphenation of words has been forbidden in Arabic since centuries. Various other methods of justification are used instead, like stretching intermediate connectors between letters and the use of an elaborate system of ligatures and alternative letter forms. The paper belonging to this talk was a joint effort with Azzeddine Lazrek and the calligrapher Mohamed Benatia, who kindly calligraphed the phonetic representation of all of our names in Arabic.

Officially, this would have been the last talk of the day, but to kick of the round table discussion that followed, Yannis Haralambous presented a set of slides showing all the various languages that use Arabic script and the enormous amount of variation that can be found therein. After that, the round table discussion followed. This was attended by a fair number of local students as well as TeX-ies.





Saturday November 11, day 3

Claudio Beccari kicked off the first session of the final day, talking about the use of pict2e, a fairly recent addition to the LaTeX repertoire of drawing packages. The focus was on how you can use complex numbers to simplify drawing.

LaTeX3, and especially page design, was highlighted by Morten Høgholm. Because he received many general questions in preceding days, he also explained quite a bit about the project infrastructure and the internals of the upcoming LaTeX3. There is still no release date, but the code is ready enough interested people are requested beta–test the new system.

After the coffee break, there was a session on fonts. It started with Jerzy Ludwichoski who presented the TeX Gyre font project. You can read all about that elsewhere in the Maps.

Macintosh users can be proud of the fact that the i-Installer already has support for the first three font families that came out of this project, because in the next slot, Gerben actually built the required i-Packages as a demonstration of how to use i-Installer 'from the other side'.

At this point, a presentation was squeezed in. Yannis Haralambous did a brave attempt to replace Apostolos Syropoulos based on his slides entitled 'LaTeX as a tool for the typographic reproduction of ancient texts'. For a large number of ancient scripts, this talk explained some of the problems related to the script and how well it can be typeset using current LaTeX. Yannis worked hard to compensate the missing author by showing examples he had found of the scripts in question.

'Everything we want to know about modern font technologies'. Chris Rowley turned his presentation into a panel session, made possible by the the presence of many font experts. He had prepared some questions and after discussing them shortly, a vivid discussion evolved. Among the topics discussed were the fundamental differences between typesetting western scripts (by pasting together glyphs) and the Arabic

script which is still strongly rooted in the calligraphic world, something not trivial to do with computers. The discussion ended with musings about the potential of using MetaPost as an integral part of TEX, enabling the use of dynamically generated fonts.

Hurray, after that it was time for lunch again! Following that, Elena Smirnova talked to us about how to generate TEX from mathematical content while honoring notational preferences. To this end, they convert from and extended form of content–MathML to presentation MathML or TEX input using a specific notation style that can be selected by the user using a simple GUI interface. The conversion application itself is controlled by an XML-based catalog of possible presentations and conversions.

'dvi2svg: Using LaTEX layout on the web' was presented by Adrian Frischauf. He convincingly demonstrated that you can create really good looking math in web pages by converting the TEX-generated DVI file into SVG, and then serving the SVG to the web browser. The biggest (perhaps only) drawback of this approach is that right now, it needs a separate plug-in to be installed in order to view the page. The built-in support inside web browsers is not yet good enough to handle complex text native.

Hans Hagen talked about the impact of luaTEX on ConTEXt. he gave an overview of the status quo of the luaTEX project and gave some examples of how and where Lua comes into play. This talk was a summary of the "MKII – MKIV" paper you can find elsewhere in this Maps issue.

After the coffee break and the traditional group photograph, the final session started with a continuation of Gyöngyi Dujdosó's talk at EuroTEX2006. 'TEX, typography & art together' focused on the typographical side of a TEX- and typographical e-learning system that is being developed in Hungary. This second side of the system revolves around a database containing typefaces, paintings, page layouts, and information about their designers.

More than twenty-five years of TUGboat history were summarized by Barbara Beeton in the final presentation of TUG2006. Barbara presented a large number of examples that were scanned in from previous volumes as an aid in explaining how the current layout came to be.

Several talks dealt with the more trendy ways of coding (xml) and presenting math (web, dynamic, right-left). In the final discussion Stephen Watt challenged the audience to come up with visions and requests concerning coding and representing math as well as current and future demands of accessing math on the web. Being a member of the forums that discuss these items in relation to standards, the author triggered discussion and also invited the audience to bombard him with emails regarding the subject.

That night was the banquet, held at a special place called 'Chez Ali'. After dinner, a special performance was held in the center courtyard of the complex. At the end of that there were fireworks and a ritual burning of the TUG logo!

Sunday November 12, excursion and closing

Even though the last talk was on Saturday, almost everybody stayed around for the excursion to Essaouira on Sunday. Perhaps in part because the banquet was the night before, so that is a trick worth remembering. After the crowded and metropolitan Marrakesh, it was nice to see a town that was perhaps just as commercial, but much, much smaller in scale.

when we got back to the hotel in the evening, the time had come to officially close the conference. Many words of thanks were spoken there already, but let me repeat again: a big thank you to the local organization and especially Azzeddine Lazrek (in the black trousers at the front right of the group photograph). It has been a great TUG conference!

Taco Hoekwater