MAPS

NUMMER 36 • VOORJAAR 2008

NTG NEDERLANDSTALIGE TEX GEBRUIKERSGROEP

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 software-archieven.
- ☐ 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 pdftex 1.40.3-2.2 (Web2C 7.5.6) 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

Deze Maps is, voorzover ik kan nagaan, de eerste waarin ConTEXt niet alleen in het aantal gezette bladzijden, maar ook in het aantal aangeleverde artikelen, een voorsprong neemt op LaTEX. Gelukkig maar, want het is ook de eerste Maps helemaal zonder Siep Kroonenberg, waardoor de LaTEX sectie van de redactie wat onderbezet is.

Al een paar jaar staat onder het kopje **Productie** in de colofon van de Maps iets als 'het pdf bestand voor de drukker wordt aangemaakt met behulp van pdftex . . . '. Voor de Maps die nu voor u ligt is dat eigenlijk een leugen, want meerdere artikelen in deze Maps zijn geproduceerd met andere TEX-varianten zoals XƎTEX en LuaTEX. We hebben dan ook een heel gevarieerd aanbod deze keer.

- Meteen al het eerste artikel in deze Maps, dat van Wilfred van Rooijen (*Typesetting CJK and other exotic characters*), is zelfs helemaal niet door de redactie getypeset. Toen de eerste versie van dat artikel binnenkwam was onmiddellijk (en pijnlijk) duidelijk dat de Maps–productie nog niet klaar is voor CJK typesetting. Wilfred was zo vriendelijk de definitieve pdf-bestanden zelf voor ons klaar te maken.
- Jelle Huisman's artikel (*Met XƎTEX meertalig*) is weliswaar op de redactie gezet, en zelfs met pdftex, maar dat kon alleen omdat Jelle alle voorbeelden had aangeleverd als pdf-bestanden. En die pdf-bestanden zijn uiteraard gegeneerd met XƎTEX.
- Een vaak gestelde beginnersvraag is 'Wat is nu het verschil tussen LaTEX en MikTEX, of tussen pdfTEX en ConTEXt?' Het artikel van Piet van Oostrum (*What is it about all those *TEXs*) schept orde in het oerwoud van termen met 'TEX' in de naam.
- Daaropvolgend kunt u twee verschillende recensies lezen van de Engelse vertaling van Yannis Haralambous' boek over Fonts en Encodings. De eerste is van de hand van Ulrik Vieth, de tweede van Luigi Scarso. Vanwege de zeer grote verschillen tussen deze twee artikelen vonden we het de moeite waard beide af te drukken.
- Hans Hagen's artikel over de 'lange ij' (*Latin Modern Nederlands*) is gezet met LuaTEX, omdat de benodigde OpenType trickery alleen in LuaTEX en XƎTEX mogelijk is, niet in pdfTEX.
- In *Theorems in ConTEXt* laat Aditya Mahajan ons zien welke recente uitbreidigen in ConTEXt zijn toegevoegd voor theorems en proeven.
- Hans van der Meer is in deze Maps vertegenwoordigd met twee artikelen. De eerste daarvan (*Exam Papers*) introduceert zijn module voor de productie en het onderhoud van examens.
- Documentbeheer is het onderwerp van het artikel van Roland Smith (*Revision control for TEX documents*).
- De nieuwste versie van ConTEXt heeft uitgebreide ondersteuning voor het gebruik van LuaTEX. Hans Hagen's tweede artikel (*The luafication of TEX and ConTEXt*) geeft een inleiding in de daardoor toegevoegde functionaliteit.
- Iets heel anders komt aan bod in de volgende twee artikelen: Frans Goddijn (*DHZ Boek*) en David Walden (*Notes on Self-publishing*) verhalen allebei over het zelf publiceren van boeken en wat daar allemaal bij komt kijken.
- De volgende twee artikelen zijn beide samenwerkingen tussen Taco en Hans over de ontwikkelingen rond MetaPost. De eerste (*MetaPost library project*) beschrijft de theorie en doelen van het MPlib project, de tweede (*The MetaPost Library*) beschrijft de toepassing daarvan in ConTEXt MkIV. Dat laatste artikel is gezet met LuaTEX.
- Nog meer van Hans en Taco, nu samen met Volker Schaa, vindt u in het voorlaatste artikel van deze Maps (*Reshaping Euler*). De Euler fonts zijn door Hermann Zapf herontworpen ter ere van Knuth's 70ste verjaardag.
- Tenslotte: *Blocks and Arrows with MetaPost* is het tweede artikel van Hans van der Meer, over het gebruik van MetaPost voor het tekenen van diagrammen.

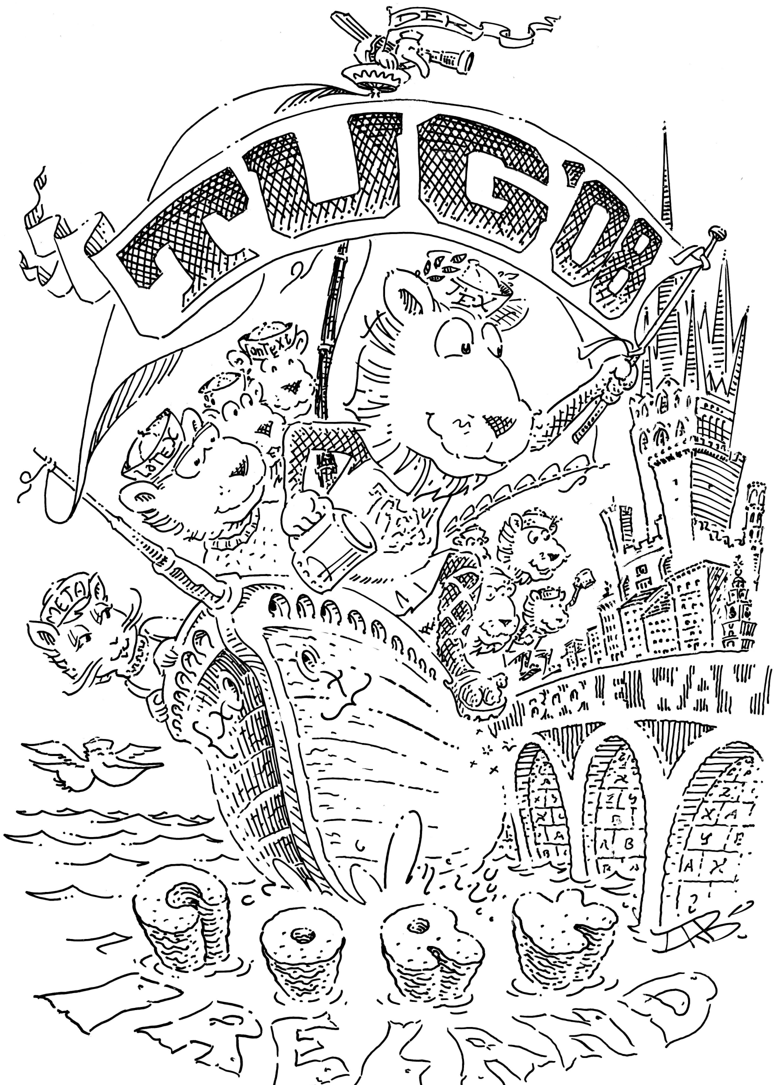
En dat was het dan weer voor deze Maps. Namens de redactie wens ik u veel leesplezier, en wellicht tot ziens op de NTG-dag in Purmerend op 26 juni.

Taco Hoekwater

TeX Users Group 2008 Conference

University College Cork
Cork, Ireland
21–24 July 2008
`http://tug2008.ucc.ie/`

TeX's 30th birthday
Interfaces to TeX
Workshops
Presentations



Hosted by the Human Factors Research Group (`http://hfrg.ucc.ie`)

Typesetting CJK and other exotic characters using LaTEX and XƎLaTEX

Anything goes (well, almost...)

Abstract
This paper tries to illustrate some of the particularities of typesetting CJK characters using several flavors of LaTEX. Special attention is given to Japanese. A short introduction is given about the nature of the character scripts and the special demands those alphabets put on character and font encodings. Typesetting Japanese using p(te)TEX, LaTEX, Lambda, and XƎLaTEX is discussed. Special discussion is given to XƎLaTEX, and the possibilities of including annotation markup and vertical typesetting in Japanese texts using XƎLaTEX. It will be shown that although typesetting vertical material is possible with XƎTEXv0.997, more development work will be needed in this area to create a dependable vertical typesetting system.

Keywords
XƎTEX, XƎLaTEX, CJK, unicode, horizontal and vertical typesetting, Japanese

This paper is the result of a question that was asked recently on the Dutch LaTEX mailing list. The question was whether and how it would be possible to typeset Japanese with LaTEX. In 2002, I did an internship in Japan, and remembered that I had to install pTEX, which was a patched version of (then) teTEX 2.2, and then some trickery was necessary to have the correct fonts show up in the final PostScript file. Thinking that life must have become easier in the meantime, I set out on a mission to see what the different flavours of LaTEX can do in scope of the CJK languages[1]. This paper is a (not very technical) summary of my experiences. The major focus is on Japanese, but examples in all CJK languages are provided. The paper will start with a bit of history to explain the origins of the Chinese character script, which provides insight into the considerable difficulties that (used to) exist in using CJK on computers. Then we'll discuss a bit about character and font encoding, which will be followed by a listing of possibilities of incorporating CJK texts into a LaTEX document. The paper will be concluded by some examples using XƎTEX which should be reproducible by anybody who has a recent version of TEXLive, Adobe Acrobat reader, and an internet browser.

History of Japanese characters

This short introduction follows that of [1]. Chinese characters originated in the Yangtze River region of China, between 2000 – 1500 BC. Starting as simple pictographs, the characters evolved to also express abstract concepts. Several pictographs could be combined into one character to express complex ideas, and provide different nuances in meaning. The well known square-formed characters (known in Japanese as kaisho, 楷書) developed around 200 AD. A more or less formalized system evolved where each character has a main part expressing the base meaning of the character (Japanese: 部首, 'radical'), adorned with other radicals to express pronunciation and nuance of meaning. It should come as no surprise that such a system can easily lead to a large number of different characters. Around 200 AD, there were an estimated 50.000 characters.

Chinese characters entered Japan between the third and fourth century AD, mainly by Chinese and Korean monks and scholars. In fact, the word kanji (漢字) literally means 'letters of the Han Dynasty' (206 BC – 220 AD). In Japan, kanji were initially only used to write Chinese texts, but over time kanji came to be used for Japanese texts as well. This lead to the development of different pronunciations for the same character. For example, the kanji 国 is pronounced 'kuni' in Japanese reading (kun yomi (訓読み), litt. 'reading for meaning'), and 'KOKU' in Chinese reading (on yomi (音読み), litt. 'reading for sound'). Usually, kanji appearing by themselves are read in kun yomi, and in combination kanji are read in on yomi: 母国 BOKOKU, 母 haha, 国 kuni.

The fundamental differences between Chinese and Japanese are that Japanese is an inflected, polysyllabic, non-tonal language, whereas Chinese is the opposite. Over time, it became necessary in order to write Japanese to come up with a way of expressing the inflected parts of verbs, for instance. From around the seventh century AD a system developed to express these inflected parts by standard kanji used phonetically: man'yōgana[2]. The man'yōgana eventually led to the kana (仮名, 'assumed names'), which are purely phonetic. The modern descendants of kana are hiragana and katakana. In modern Japanese, the non-inflectional part of a verb is written with kanji, and the inflected part is written in hiragana: 見る 'to see', 見ない 'to not see'. Katakana is used to write foreign words in Japanese transcription: ソースコード 'source code'.

In Chinese it is possible to distiguish between homophonic kanji by their tonality, but in Japanese that is not possible. As a result, many different kanji obtained an identical pronunciation. Over time in China the pronunciation of standard Chinese changed. The Japanese incorporated many of the 'newer' pronunciations of the existing kanji into their vocabulary, so that finally each character may have many different pronunciations (examples are 下, 'below', which has ten pronunciations, and 生 'life', with nine), and there are many characters sharing the same pronunciation. My electronic dictionary lists 323 kanji with pronunciation 'kō', and 267 under 'shō'. Because there are so many homophonic kanji, many television programs are subtitled.

Much debate rages over the total number of kanji. The famous Dai Kan-Wa Jiten (大漢和辞典)[3] 'Great Chinese - Japanese Dictionary'), published since 1955, contains a total of 49.964 kanji (although most of those differ only in their radicals). After 1945, the Japanese ministry of education tried to standardize a list of kanji and produced the 'tōyō kanji' list, litt. 'temporary use kanji', with 1850 kanji, 881 of which are known as 'kyōiku kanji' which are taught in the first six years of school. In 1981, the list was revised to become the 'jōyō kanji' list (general use kanji) with 1945 kanji, 996 of which are taught. Adherence to this list is not strictly enforced, and especially in scholarly works, literature and poetry many non-jōyō kanji can be found. For example, Kodansha's essential kanji dictionary [2] lists 1945 kanji, while the Compact Nelson [3] lists 3068 kanji, and my electronic dictionary lists 6355.

Traditional and simplified Chinese

In China simplified characters have been used for many centuries, but those were not used in print widely. From 1949, the Communist government began assembling official lists of simplified kanji for everyday use in print. The current official list of simplified Chinese contains 2249 characters. This list is also the official list for Singapore and Malaysia. However, in the parts of China not influenced by communism (Taiwan, Hong Kong and Macau) the traditional characters have been in use continuously, of which no definitive number exist. As an illustration of simplified resp. traditional characters, consider the words 'China' and 'university': 中国 vs. 中國, 大学 vs. 大學.

Korea: hangul, chosŏn'gŭl and hanja

In Korea Chinese characters were also introduced very early on. The typical hangul alphabet (한글) was officially introduced in 1443 by King Sejong the Great. In its base form, hangul is a purely phonetic alphabet, with each symbol representing one sound. However, this system is complicated by the fact that several symbols can be assembled into one character to represent one syllabic block. For example, the word 'hangul' is composed of two syllabic blocks 한 + 글, which are each composed of three symbols: 한 + 글 = 'ㅎㅏㄴ' + 'ㄱㅡㄹ', 'h' 'a' 'n' + 'g' 'u' 'l'. There are 11.172 valid combinations in hangul. Apart from hangul, Chinese characters, known as hanja, are also still used on a small scale, for instance in proper names, official paperwork etc. In North-Korea the same hangul alphabet is used, although it is called choso˘n'gŭl (조선글), and no hanja are used.

Character encoding and font encoding

A computer can only handle information in the form of a stream of bits, and thus for a computer to handle characters, one needs a one-to-one mapping, mapping each character to a unique numerical representation. This numerical value is subsequently transformed into a sequence of bits in a prescribed manner. The number of characters that can be encoded depends on the number of bits that is used for the mapping (encoding). Traditional ASCII uses 7 bits for encoding, allowing a total of 128 possible characters, which is too limited to express even the simplest character lists in the CJK languages. For this reason, different encoding schemes were developed for CJK. For

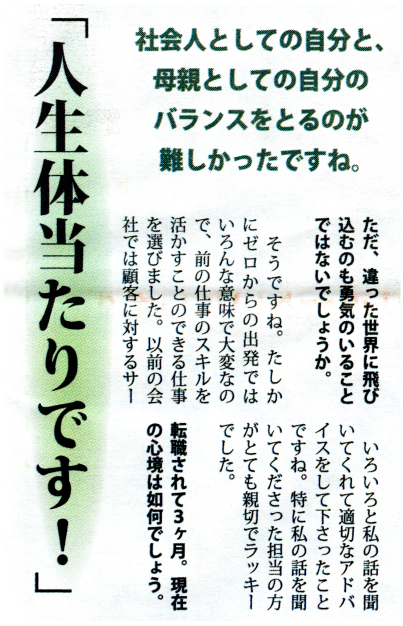
Japanese, EUC-JP, JIS and SJIS were developed. EUC (Extended Unix Code) is a multi-byte encoding; each character is encoded using either 1, 2 or 3 bytes, and each byte is capable of representing 94 characters (several bits per byte are required to distinguish whether the byte is part of a one-, two- or three-byte character, and hence not all bits are available to encode characters). Besides EUC-JP, there are EUC-CN (Chinese), EUC-KR (Korean) and EUC-TW (Taiwanese). EUC-JP is the norm for Unix(-like) operating systems.

JIS (Japanese Industrial Standards) consists of 6879 kanji and a specification for encoding into one or two bytes. A maximum of $94 \times 94 = 8836$ positions are available in JIS. A competing encoding is Shift-JIS (SJIS), originally developed by Microsoft (and others). SJIS differs from JIS in how the numerical values of characters are translated to one or two bytes. Because of the nature of SJIS, it is difficult to detect SJIS encoding automatically, often resulting in a messy screen, known in Japanese as 文字化け (mojibake), 'characters in disguise'. Individual vendors use the space not taken by the JIS character set to add their own characters to SJIS. For example, mobile phone operators use this space to encode emoticons, amongst other things. Microsoft uses their own extended SJIS in Windows (Codepage 932)[4].

Unicode[5] is an encoding standard with enough room to encode millions of different characters in one large set of assigned code points. Unicode provides three different ways to translate characters code points into a form capable of transmission: UTF-8 (Unicode Transformation Format 8) translates the code points into 8 bit units: a sequence of 1, 2, 3 or 4 bytes; UTF-16 translates into 16 bit units; and UTF-32 translates into 32-bit units. The total number of possible code points is 1114112 ($2^{20} + 2^{16}$).

Other encodings, unicode and han unification

For Chinese, Taiwanese, Hong Kong-ese, and Korean different encoding schemes were developed (Big5 for traditional Chinese, BG for mainland Chinese, KS for Korean). This makes it virtually impossible to typeset more than one of the CJK languages within the same file, because different characters would resolve to the same numerical presentation, and it would depend on the font encoding which character is actually displayed. Only unicode encodes all the characters separately, making it possible to use all kinds of alphabets indiscriminately in the same file. To reduce the number of CJK characters in unicode, the so-called 'han unification' is implemented, where several variants of the same character common to the CJK languages are mapped to the same unicode position. This leads to occasional protest, for instance when a character historically used for a proper name is designated as a variant of another character. The specific variant can then no longer be encoded separately in unicode, and cannot be typeset on a computer.



社会人としての自分と、
母親としての自分の
バランスをとるのが
難しかったですね。

ただ、違った世界に飛び込むのも勇気のいることではないでしょうか。

そうですね。たしかにゼロからの出発ではいろんな意味で大変なので、前の仕事のスキルを活かすことのできる仕事を選びました。以前の会社では顧客に対するサー

いろいろと私の話を聞いてくれて適切なアドバイスをして下さったことですね。特に私の話を聞いてくださった担当の方がとても親切でラッキーでした。

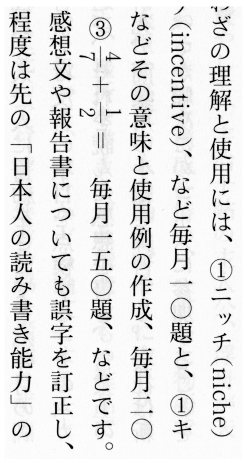
転職されて3ヶ月。現在の心境は如何でしょう。

「人生体当たりです！」

Figure 1. An example showing the combined use of horizontal and vertical typesetting in a Japanese news paper article.

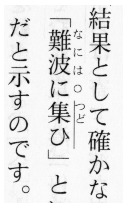
LaTEX and CJK

To typeset a text, the computer will read the input stream, and interprets a given sequence of bits as representing a certain character, based on the character encoding used. The corresponding character in the font set should then be displayed. If one has a font with the same font encoding as the input encoding, this implies a one-to-one mapping. If on the other hand a unicode font set is used with, say, SJIS encoding of the file, the SJIS characters from the input stream have to be translated to unicode values in order to display the correct character on the screen using the unicode font. Traditional LaTEX has several problems here because of built-in limitations: EUC or (S)JIS input has to be read, and a translation provided to typeset the cor-



ざの理解と使用には、①ニッチ(niche)
(incentive)、など毎月一〇題と、①キ
などその意味と使用例の作成、毎月二〇
③ $\frac{4}{7}+\frac{1}{2}=$ 毎月一五〇題、などです。
感想文や報告書についても誤字を訂正し、
程度は先の「日本人の読み書き能力」の

Figure 2. An example showing various features of Japanese typesetting: Latin text is set sideways in vertical typesetting. The `equation' is also set sideways. The two punctuation marks and are set in burasage.



結果として確かな
「$\overset{\text{なには}}{\text{難波}}$。に$\overset{\text{つど}}{\text{集}}$ひ」と、
だと示すのです。

Figure 3. An example of ruby, or furigana. The first two characters are glossed for pronunciation. The punctuation mark marks a grammatical rule. The last ruby denote a highly uncommon pronunciation.

rect characters at the correct location, and the resulting DVI stream has to be correctly translated to a PS file (provided that an adequate PS font is available with glyphs for the kanji). Legacy TEX only allows for 256 character (1-byte) encodings. Several patches and packages have been developed over the years to circumvent these problems, as will be discussed later.

Specific typographic rules exist for CJK. In the case of Japanese, texts can be either read horizontally from left to right (LTR), or vertically from right to left (RTL). Almost all printed material in Japanese is set vertically: news papers, magazines, manga etc. Publications in the 'hard' sciences are usually set horizontally because of the presence of equations. Advertisements in printed matter, tabloids etc commonly feature both horizontal and vertical typesetting. Legacy TEX will only allow LTR typesetting. e-TEX allows LTR and RTL. XƎTEX is the only flavor capable of setting texts vertically (although this depends on the specifics of the font used, as will be shown later). In figure 1 an illustration provided showing mixed use of horizontal and vertical typesetting in one news paper article.

A common misconception is that CJK languages do not have kerning. Although it is essentially true that all characters are thought of as being written on a square grid, there are characters that do not necessarily occupy a full character position (the punctuation marks「,」,。and 、for instance). Also, some characters are 'denser' than other characters and need a bit more room around them for legibility[6]. Most CJK fonts do not support kerning, but professional DTP software does. Another specific feature is that punctuation marks are allowed to protrude into the margins (burasage). See figure 2 for an illustration.

Another feature of Japanese is 'ruby': hiragana characters printed above kanji (in horizontal texts) or to the right of kanji (in vertical texts) to indicate pronunciation[7] as illustrated in figure 3. Of course, ruby is most commonly encountered in publications for young readers, but you see it occasionally on name tags and official paperwork.

Japanese fonts are usually available as Gothic and Mincho. Gothic is comparable to sans-serif, mincho is comparable to a serif font, and is most commonly used for printing. For manga etc, a 'hand-written' style is usual, while for poetry the 'cursive' style is commonly used. In cursive, the brush does not leave the paper when writing a character, yielding highly stylized and abstracted characters (in extreme cases denoted as 'grassy'). Please refer to the figures for some illustrations of the different styles of kanji.

漢字は難しい。時々全然読めない。

Figure 4. Mincho typeface (Kozuka Mincho Pro-VI)

漢字は難しい。時々全然読めない。

Figure 5. Gothic typeface (Kozuka Gothic Pro)

漢字は難しい。 時々全然読めない。

Figure 6. Manga style hand-written typeface (YOzFontN04)

漢字は難しい。時々全然読めない。

Figure 7. Cursive typeface (HakusyuSeigyosyoKyo)

漢字は難しい。時々全然読めない。

Figure 8. Highly cursive, `grassy' typeface (HakusyuSousyuKyo)

国会議事堂前

Figure 9. Typeface `tensho', used on seals and stamps (HakusyuTensyoKyokan). Note that even for Japanese this type of writing is not always easy to read (国会議事堂前).

Integrating CJK characters into LaTEX documents

As it turns out, there are several ways of incorporating CJK into LaTEX documents, and all of these have their strong and weak points. This part of the paper is really what the original question was all about: "I want to include some Japanese text in my LaTEX document, how do I achieve this?". And as is to be expected, not all methods are equally applicable for given circumstances. I hope to give some insight into the various possibilities, and provide some guidance to when to use which method. The discussion of XƎTEX's capabilities will be put off until the next section.

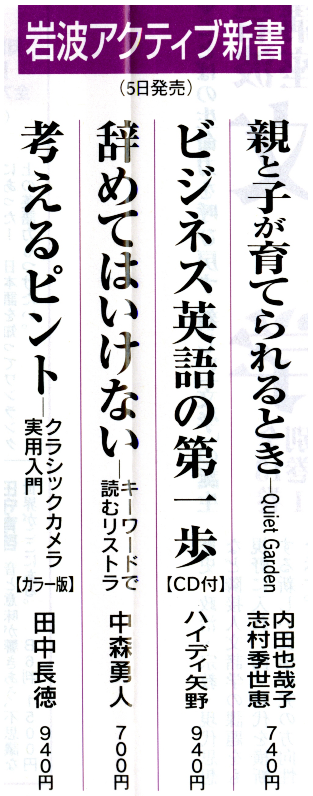
pTEX, pteTEX, and pTEXlive

In Japan a patched version of TEX was developed called pTEX. pTEX is capable of reading and typesetting EUC-JP, (S)JIS and UTF-8 encoded files. Nowadays, inclusion of the patches is automated in the pteTEX and pTEXlive distributions[8]. Patching is required of TEX and dvips, although nowadays only patches for dvipdfmx are available. The user is required to install an adequate font for Japanese, like Cyberbit or Sazanami.

The pTEX distributions are the most complete way of typesetting Japanese with LaTEX, supporting burasage, ruby, and full vertical typesetting. The drawback is that only Japanese can be typeset. pteTEX is included in many linux distributions for the Japanese market, like VineLinux and TurboLinux, and pTEXLive is available as a small set of patches whose inclusion into a regular TEXLive distribution is automated.

Traditional LaTEX and the CJK package

Another option is to use the CJK package, available in TEXlive 2007. This package allows typesetting of Chinese, Japanese, Korean and Thai. The CJK package uses a pre-processor based on the Mule package of XEmacs (`cjk-enc.el`) to translate an in-


岩波アクティブ新書
（5日発売）

親と子が育てられるとき—Quiet Garden 内田也哉子 志村季世恵 740円

ビジネス英語の第一歩【CD付】 ハイディ矢野 940円

辞めてはいけない—キーワードで読むリストラ 中森勇人 700円

考えるピント—クラシックカメラ実用入門【カラー版】 田中長徳 940円

Figure 10. Example showing gothic and cursive kanji styles used together. This is from a leaflet advertising new books to be published. Notice that contrary to popular belief latin numerals can be used in vertical typesetting. Also notice that horizontal and vertical typesetting are often mixed.

put file in a given encoding (EUC-JP, Big5, BG, ...) to some canonical form, and then heavily uses translations from the input encoding to font encoding to typeset all the characters. Because the CJK package functions within legacy LaTEX, the font encoding (in NFSS) plays an important role. If only one of the CJK languages is used, the input file can be encoded in a relevant encoding (SJIS for Japanese, for instance), and pre-processed 'on the fly' using the `sjis(pdf)latex, bg5(pdf)latex, ...` scripts which are available in TEXlive 2007. If more than one CJK language is used in one file, the file should be encoded in UTF-8, preprocessed with (X)Emacs Mule, and then through LaTEX. Switching between the languages and the corresponding font families has to be done by the user using the corresponding commands in the input file. Because of the way the CJK package typesets the material, it is quite slow, and according to the manual, should only be used to typeset some CJK material in a given document, but is not very efficient to typeset large documents in the CJK languages. Personally, I consider the CJK package to be very useful (it does fully support ruby, for instance), but only relevant for the somewhat advanced user, mainly because of the selection of relevant font families (leading to all kinds of issues with .fd and .map files).

Traditional LaTEX and the UCS package

Legacy LaTEX is perfectly capable of reading unicode input files with the UCS package. However, as discussed above, being capable of reading an input stream and subsequently putting the correct character in the output are two different things. If LaTEX is to typeset a Japanese unicode input, a translation still has to be made from the unicode input into some font encoding for NFSS to access the glyphs. For example, to properly typeset some Japanese text, \usepackage[utf8x] {inputenc} and \usepackage[C42,T1]{fontenc} are required to instruct LaTEX to read UTF-8 input, translate to C42 (NFSS SJIS) for the CJK characters, and use T1 for the non-CJK characters (the legacy CJK fonts in LaTEX do usually not include the 'latin' part of the font set, so a 'T1-capable' font like latin modern is used instead). If the options for fontenc are set correctly, TEXLive 2007 will run correctly (but note that if C40 is chosen instead of C42, the required kanji fonts are not included in TEXLive).

After some experimenting with UCS, I found that it will only typeset one of the CJK languages in a given document. I also found that line breaking is not performed, because legacy LaTEX uses whitespace or hyphenation for points where a line can be broken, neither of which are present in a CJK text. Ruby is not supported, although a package could probably be written. The UCS manuals and documentation are very sparse.

Omega, Lambda and dvipdfmx

My next attempt was to use Ω (Omega, an extended version of TEX) and Λ (Lambda, 'latex for omega') to typeset CJK material. Lambda can read unicode input and typeset LTR and RTL languages. Native Lambda does not support the CJK languages, so a patch (Omega-j) is needed[9]. For the CJK languages, the resulting DVI file can be converted to PDF with dvipdfmx, but only after some tweaking (see http://oku.edu.mie-u.ac.jp/~okumura/texfaq/japanese/ for details). Technically, Lambda and Omega have the possibilities of fully supporting Japanese, including vertical typesetting and burasage, although I was not capable of reproducing the vertical typesetting example I found in one of the manuals, and setting up the burasage requires manual tweaking of OTP files (which I consider to be too much hassle). Ruby is not supported, although a package could probably be written. But perhaps the biggest drawback is that Lambda and Omega are no longer being developed.

XƎTEX and XƎLaTEX: anything goes!

The most natural way of typesetting each and any character with LaTEX would be to use unicode encoding for the input, and a unicode encoded font for the output. This is what XƎTEX[10] is capable of doing. XƎTEX is an extension of TEX, and was originally written specifically for Mac OS. It is currently available on linux (TEXlive 2007, v0.996) and Windows (MikTEX, W32TEX, v0.997). XƎLaTEX is latex for XƎTEX. The two strong points of XƎTEX are that it is fully capable of handling unicode input, and it interacts directly with (unicode encoded) OpenType fonts installed on the computer. The direct interaction with the OpenType fonts installed on the system means that if one has a CJK-capable font available, typesetting CJK in XƎLaTEX becomes very easy: simply type the text into your favorite editor, run `xelatex` and behold the result. The direct interaction with the system fonts also implies that some of the finer details of typesetting are taken out of TEX, and are instead left to the peculiarities of the font in question and the font rendering software available on the system. While this may be unacceptable to the professional typesetter, it is a major improvement for the (advanced) LaTEX user, because there is no longer a need to deal with setting up all those .tfm, .fd and .map files that make font selection in LaTEX a hassle.

To use XƎTEX to typeset any type of character, one only needs:

1. An editor capable of reading and writing UTF-8 encoded files (for most recent linux distributions, UTF-8 is the default encoding, and gedit, XEmacs and other editors support it)
2. An OpenType font which has glyphs for the particular characters you want to appear in the output.

Since XƎTEX reads unicode directly and uses unicode encoded fonts, an input file to typeset Japanese in XƎLaTEX could be as simple as the following example:

```
\documentclass[]{article}
\usepackage{fontspec}
\begin{document}
\fontspec[Mapping=tex-text]{Kozuka Mincho Pro-
VI R}
Kozuka Mincho Pro-VI R: This is English. これは日本語で
す。
\fontspec[Mapping=tex-text]{Sazanami Mincho}
Sazanami Mincho: This is English. これは日本語です。
\end{document}
```

In this example, the `fontspec` package is used. This package provides a very simple interface to select a particular OpenType font for the document. It also provides options to use special font features. Note that this example shows that it is not necessary to load any other packages to typeset the Japanese characters if this input file is saved in UTF-8 encoding.

Prerequisites to use exotic characters in XeLaTeX

To typeset a particular text using 'exotic' characters, the first thing that is needed is an OpenType font with glyphs for the characters you wish to obtain in the output. For Japanese, a good option is the Adobe Acrobat Reader Japanese Language pack[11]. This will install two .otf fonts, Kozuka Gothic and Kozuka Mincho (on linux systems, the files are named KozGoPro-Medium.otf and KozMinProVI-Regular.otf). Another option is the Cyberbit font[12], or the Sazanami fonts[13].

To install extra fonts on a (recent) linux system, simply put the (.otf, .ttf) file in $HOME/.fonts and run `fc-cache -fv` to update the system font database. To use a given font in your document, you need to know the name of the font to enter in the \fontspec{} command. A list of available fonts on your linux system can be obtained by running `fc-list`, which will give a list of font names and capabilities. For example, running '`fc-list | grep Koz`' yields:

- Kozuka Mincho ProVI, 小塚明朝 Pro-VI,Kozuka Mincho ProVI R,小塚明朝 ProVI R:style=R,Regular
- Kozuka Gothic Pro, 小塚ゴシック Pro,Kozuka Gothic Pro M,小塚ゴシック Pro M:style=M,Regular

while '`fc-list | grep Cyber`' results in

- Bitstream Cyberbit:style=Roman

The fonts in the Acrobat Reader Japanese Language Pack are not by default stored in a systemwide font directory. I have copied the fonts to my $HOME/.fonts directory for use in this paper. Recent installations of OpenOffice provide a.o. the Baekmuk fonts (for Korean) and the AR (Arphic) family of Chinese fonts. Free OpenType fonts are available for many languages, and an internet search will reveal candidate fonts for your language of choice. Unicode supports many contemporary languages, as well as other scripts, like medieval variant alphabets, Ancient Greek Linear-B, and cuneiform etc. All these can be set with XƎLaTEX if one has an adequate font available. Especially useful fonts are Code2000 and Code2001, which have glyphs for many character sets, for example cuneiform using Code2001: [illegible]. For languages not endorsed in the unicode standard, a 'private range' is left available in unicode for individual use; candidates for the private range are for instance Klingon ([illegible]) and Elvish (Tengwar): [illegible], set with Code 2000). Note that line breaking etc is not (yet) properly defined for these languages, hence underfull and overfull hboxes result.

To enter Japanese or Chinese into a computer requires a special input method. This kind of software will not be decribed here in detail. For Windows, the IME (Input Method Editor) allows switching between latin and Japanese input. On linux, canna and SCIM or UIM do the same. As a subsitute, go to `http://babel.altavista.com/` and type in some words, then have it translated to the character types you like to try (Japanese, Simplified or Traditional Chinese, Korean, as long as your font is capable of displaying the characters). Copy-paste the result in your editor, save as UTF-8, run `xelatex`, and enjoy your PDF output.

Specific support for Japanese typography

In XƎTEX v0.996 there is no specific support for the Japanese language. Most notably, support for ruby is lacking. With a simple patch, the existing package `nruby.sty` allows simple ruby support. Burasage is not (yet) supported. Babel support is not (yet) fully available for the CJK languages. Bibtex seems to work with UTF-8 input files with CJK characters (the reference list for this paper is made with bibtex).

For XƎTEX v0.997 (available through `svn`) the package `zhspacing.sty` is under development to bring specific support for inter-character spacing in CJK, line breaks in CJK texts, spacing between CJK and non-CJK text, and support for CJK characters as elements in mathematical equations (superscript, subscript etc).

Support for vertical typesetting

XƎTEX does not support vertical typesetting per se, but it is still possible to typeset material vertically. To achieve this, one has to have an OpenType font with the 'vrt2' property. Glyphs in such a font can be rotated. If one puts some CJK text with rotated glyphs inside a \rotatebox, vertical typesetting is obtained. It should be noted that vertical typesetting is not very stable at this moment. Several manuals give examples of vertical typesetting (e.g. the fontspec and zhspacing documentation), but Your Mileage May Vary depending on your system. In my case, Texlive 2007 with XƎTEX v0.996 yielded incorrect typesetting in vertical mode.

I was advised to upgrade to v0.997, because of better support for vertical typesetting. After some trial and error, I was able to typeset some material vertically, as illustrated in figure 11, which was set with the following source:

```
\begin{figure}
\begin{center}
\rotatebox{-90}{
\begin{minipage}{0.35\textwidth}

\fontspec[Script=CJK,RawFeature=vertical]{Kozuka Mincho Pro-VI}
三一 \ruby{坂上是則}{さかのうえのこれのり}
\\[2\baselineskip]

\fontspec[Script=CJK,RawFeature=vertical]
{YOzFontKA04}
朝ぼらけ

\\[0.5\baselineskip]
\ruby{有明}{ありあけ}の月と　見るまでに

\\[0.5\baselineskip]
\ruby{吉野}{よしの}の里に　ふれる\ruby{白雪}{しらゆき}

\end{minipage}
}
\end{center}
\end{figure}
```

To obtain the result of figure 11, the text is set in a minipage with rotated glyphs. The entire minipage is put inside a \rotatebox. The ruby is provided by the `nruby` package. The line spacing is not optimal with ruby, so some extra space is added

三一　坂上是則(さかのうえのこれのり)

朝ぼらけ

有明(ありあけ)の月と　見るまでに

吉野(よしの)の里に　ふれる白雪(しらゆき)

Figure 11. From the `Hyakunin Isshu', the `Hundred poems by the famous poets' [4], Poem 31, by Sakanoue no Korenori: *At the firsrt light, it is not really the late moon shining, that casts its light on Yoshino, but the whiteness of the snow.* Set with YOzFontKA04.

manually here. Without the minipage, incorrect line breaking would occur. XƎTEX has some rudimentary support built in for line breaking in CJK languages using the \XeTeXlinebreaklocale="ja" command, but to get an acceptable result the package `zhspacing` should be used. Also, the combination of a rotatebox and minipage implies that sectioning commands are not available, and if more than one page of text has to be set, a simple overfull box will result, instead of the text being set on the next page. At the time of writing, a discussion was going on as to the best solution to this problem.

Some more examples of XƎLaTEX capabilities

Here follows some material in Hindi, Chinese, Vietnamese, and Japanese.

Hindi, using Raghindi font (raghu.ttf)

अमरीकी में मंदी की आशंका को देखते हुए अमरीकी केंद्रीय बैंक फेडरल रिज़र्व ने ब्याज दरों में आधे फीसदी की कटौती की है. फेडरल रिज़र्व ने दो दिनों की बैठक के बाद ब्याज दरों को 3.5 फीसदी से घटाकर तीन फीसदी कर दिया है. पिछले सप्ताह ही केंद्रीय बैंक ने दुनिया के शेयर बाज़ारों को संभालने के लिए ब्याज दरों में कटौती की घोषणा की थी. माना जा रहा है कि इस कदम से अमरीकी अर्थव्यवस्था को मंदी की संभावना से उबरने में

मदद मिलेगी. इसके पहले 2007 के अंतिम तीन महीनों में अमरीकी अर्थव्यवस्था के आँकड़े जारी किए गए थे जिसमें अर्थव्यवस्था की मंदी के संकेत मिले थे. अक्तूबर और दिसंबर के बीच अमरीकी अर्थव्यवस्था की विकास की दर में 0.6 फ़ीसदी की गिरावट आई. विशेष क़दम विशेषज्ञों का कहना है कि फ़ेडरल रिज़र्व ने ये विशेष क़दम इसलिए उठाया है ताकि जल्द ही फिर कटौती की कोई ज़रूरत न रहे.

Chinese, AR PL ShanHeiSun Uni font
美国联邦储备局9天内第二次削减利率，试图避免美国经济进入衰退。美国央行经过两天的会议后将利率从3.5%降到3% 。上周美国联邦储备局削减利率，那次降息成为25年以来削减幅度最大的一次。美联邦储备局试图通过大幅度降息平息全球股市震荡。美国联邦储备局希望降息能够减少信贷紧缩和房屋市场衰退对美国经济的冲击。美国央行的联邦公开市场委员会(FOMC)表示，金融市场仍然受到很大压力，商业和家庭信贷会进一步紧缩 ，另外最新的数据显示房屋市场进一步收缩，劳务市场出现疲软。股市上扬降息半个百分点已经超过了一些经济学家的预测，所以降息会鼓舞金融市场。

Vietnamese, Bitstream Cyberbit font
Ngân Hàng Trung Ương Hoa Kỳ đã cắt giảm lãi xuất chính nửa phần trăm để xuống còn là 3% trong một nỗ lực nhằm vực dậy nền kinh tế của nước Mỹ mà hiện đang rơi vào một cơn đình đốn. Tổng thống Bush nói rằng "nền kinh tế của Hoa Kỳ đang phải giáp mặt với các khó khăn ngắn hạn, và ông vững tin vào triển vọng xán lạn về lâu về dài". Đây là lần thứ nhì trong vòng tám ngày, Ngân Hàng Trung Ương đã cắt lãi xuất và cũng là lần đầu tiên từ 25 năm nay, Ngân Hàng mới áp dụng một biện pháp nhanh gọn đến như thế.

Korean, Baekmuk Gulim font
‘단군 이래 최대 소송’으로 일컬어지는 삼성자동차 채권 환수 소송에서 삼성측이 약 3조1500억원을 물어내라는 판결이 내려졌다.서울중앙지법 민사합의21부(김재복 부장판사)는 31일 삼성자동차 채권단인 서울보증보험 등 14개 금융기관이이건희 회장과 삼성그룹의 28개 계열사를 상대로 낸 약 5조원의 약정금 청구 소송에서 원고 일부 승소 판결을 내렸다.재 판부는 “채권단의 주장에 상당부분 일리가 있다”며 “삼성측은 채권단과 약정한 2조 4500억원을 모두 갚아야 한다”고 판결했다. 재판부는 “채권단이 맡고 있는 주식을 삼성측이 팔아서 1조 6338억여원까지 만들고 나머지 부족한 부분은 이건희 회장이 개인적으로 갖고 있는 삼성생명 주식을 팔아서 2조 4500억원을 채우라”고 판결했다. 재판부는 그러나 연체이자는 1조 6338억여원에 해당하는 이자만 지급하면 된다고 판결했다. 이자가 2001년부터 계산되기 때문에 약 7000억원에 달한다. 따라서 삼성은 원금과 이자를 합쳐 약 3조 1500억원을 채권단에 지급해야 할 것으로 보인다.

Japanese, with vertical typesetting

歴史的寒波、影響は1億人超に温首相も動く　２００８年０１月３０日１９時５５分　中国の中南部を襲った歴史的な寒波で中国政府は３０日、影響を被った人が1億人を超えたことを明らかにした。旧正月（2月7日＝春節）の帰省ラッシュを直撃した災害に政府は危機感を強め、温家宝（ウェン・チアパオ）首相を湖南、広東両省に急派した。石炭が運べず、火力発電所が広範囲で停止。日系企業にも影響が出ている。胡錦濤（フー・チンタオ）主席は２９日、輸送と発電の復旧に全力を挙げるよう指示。温首相は同日朝、被害が大きい湖南省の長沙駅で帰省客に「復旧にそれほど時間はかからない。家で年越しできる」と拡声機で励ました。　３０日は広州市の農産物市場で販売員に「値上がりしていませんか」と質問した。白菜は数日で7割上昇、羊肉も2日で5割値上がりしている。　その直前に訪ねた広州駅周辺では８０万人が足止め。当局は出稼ぎ労働者２６００万人に、「帰省せず、広東で年越しを」と呼び掛ける一方、「足止め客に宿舎と食事を確保する」と強調。上海市、福建省なども同様の状況だ。　上海では３０日まで6日間連続で雪が降った。空の便は国際、国内線で計１０００便以上が欠航、遅延。長距離列車も連日５０～６０本が運休か遅延している。主な高速道も閉鎖、上海は一時「陸の孤島」状態になった。　上海駅前では２９日午後、暖かい構内に入ろうとする客と警官がもみ合いに。店員、孫海蓉さん（１６）は「上海のアパートを引き払ったので列車に乗れないと泊まる場所もない」と話した。　日系企業が集中する江蘇省無錫市などでは、発電用の石炭不足による電力供給カットで工場の生産に支障が出た。広州でも、部品が届かず、日系乗用車メーカー３社に影響が出ている。

Concluding remarks

There are several ways of incorporating CJK material in a LaTEX document. The most complete support for typesetting Japanese, including all bells and whistles for that language, are found in the pTEXLive distribution, which specifically supports Japanese. To include a 'small amount' of CJK material in a LaTEX document, there is the `CJK` package to do all three CJK languages, with all bells and whistles. The `UCS` package provides a theoretical possibility of incorporating CJK by enabling LaTEX to read unicode input, but restrictions apply (only one CJK language per document), and without extra definitions of line breaking etc., application is limited and troublesome. One could use Lambda and Omega, but the versions supplied in TEXLive 2007 are not CJK capable and need to be patched. Most importantly, Ω is no longer being developed further, making this an unattractive option.

The most user-friendly option I found is XƎTEX and the associated macro-package XƎLaTEX in combination with `fontspec`. As long as one has an OpenType font available on the system with the appropriate glyphs, character sets can be used indiscriminately in one document, as long as glyphs are available in the font. The current version of XƎTEX on TEXLive 2007 is v0.996, and not all bells and whistles work equally well on all systems, as I found out. CJK typesetting is improved in v0.997, although some issues remain for vertical typesetting.

References

[1] K.G. Henshall. A guide to remembering Japanese characters. Charles E. Tuttle, Co., thirteenth edition, 1998.

[2] Kodansha's essential kanji dictionary. Kodansha international, 1991, 2002.

[3] The compact Nelson Japanese - English character dictionary. Charles E. Tuttle, Co., 1999, 2004.

[4] 出井光哉. プラス英訳百人一首. 風塵社, 1991.

Fonts used in this document

Various fonts are used throughout this article. All these fonts are freely available on the internet. To enable the reader to experiment, these are the font names and where they can be found. I downloaded all these fonts and installed them in $HOME/.fonts, and used 'fc-cache -fv' to make them available in XeLaTeX.

- The main font throughout is 'Kozuka Mincho Pro-VI R' (Acrobat Japanese Language Package)
- The 'YOzFont' fonts are available at http://yozvox.web.infoseek.co.jp/446F776E6C6F6164.html
- The 'Hakusyu' fonts are available at http://www.linkclub.or.jp/~ma3ki/lc-hp/font.html
- Code2000 and Code2001 are available at http://www.code2000.net
- The 'Raghindi' font used for Hindi can be found at http://tdil.mit.gov.in/download/Raghu.htm
- For Korean, the Baekmuk Gulim font is part of OpenOffice; for Chinese, the AR PL ShanHeiSun Uni is part of OpenOffice.

For the reader wanting to experiment with the possibilities of XƎLaTEXand CJK, some simple input files are available on the NTG website (go to http://www.ntg.nl/maps/36/xetex/).

Footnotes

1. Chinese, Japanese, Korean
2. Named after the man'yōshū, "Collection of Ten Thousand Leaves", a collection of poems written between 600 and 759 AD in standardized, phonetic kanji
3. http://www.taishukan.co.jp/kanji/daikanwa.html
4. I don't know whether this applies to XP / Vista
5. http://www.unicode.org/
6. See for an example http:// www.lukew.com/ ff/ entry.asp?111
7. Ruby is also known as furigana, 'guiding kana', and kanbun, 'kana letters'
8. http://www.nn.iij4u.or.jp/tutimura/
9. See http:// zoonek.free.fr/ LaTeX/ Omega-Japanese/ doc.html for an example
10. http://scripts.sil.org/xetex
11. http://www.adobe.com/products/acrobat/acrrasianfontpack.html
12. http://http.netscape.com.edgesuite.net/pub/communicator/extras/fonts/windows/
13. http://sourceforge.jp/projects/efont/files/

Wilfred van Rooijen
`wvanrooijen@yahoo.com`

Met XƎTEX meertalig
Talen en fonts in TEX

Abstract
Dit artikel is een bewerking van de lezing die ik op de NTG voorjaarsbijeenkomst van 8 juni 2007 heb gehouden. Dit artikel begint met wat achtergrondinformatie over talen, schriften en fonts. Het tweede deel van het artikel geeft een aantal voorbeelden van meertalig TEX-gebruik met behulp van XƎTEX.

Keywords
talen, schrift, fonts, XƎTEX

Talen

Als we TEX gebruiken doen we dat voor het overbrengen van informatie die in een bepaalde code weergegeven is. Deze code (meestal noemen we die code 'taal') omvat zowel de natuurlijke taal die we als mensen in het sociale verkeer gebruiken, als de kunstmatige, specialistische talen die bijvoorbeeld wiskundigen gebruiken om een n-dimensionale ruimte mee te beschrijven. In dit artikel beperk ik mij tot natuurlijke, levende talen, de talen die door mensen op dit moment gesproken worden. Hoeveel talen zijn er? Ik ga gemakshalve voorbij aan de lastige definitiekwestie van taal versus dialect en aan taalvariatie die het tellen van talen soms lastig maakt. Volgens de Ethnologue[1] worden er op dit moment ongeveer 6912 talen gesproken. Het woord 'ongeveer' staat voor: het aantal is niet precies want er worden nog steeds nieuwe talen ontdekt, terwijl er ook talen uitsterven. Dat er toch een getal staat komt omdat de gegevens uit een database komen waarin nu eenmaal 6912 records zitten van evenzoveel talen waarvan in elk geval een minimum aantal gegevens beschikbaar zijn. Een van de dingen die het meeste opvalt als het gaat om de aantallen is de enorme onbalans tussen de hoeveelheid talen en de hoeveelheid sprekers van die talen. Er zijn 10 talen met meer dan 100 miljoen sprekers: Chinees, Spaans, Engels, Arabisch, Hindi, Portugees, Bengaals, Russisch, Japans en Duits. In totaal zijn er 347 talen (ongeveer 5% van het totale aantal talen) met meer dan één miljoen sprekers. Deze talen worden gesproken door 94% van de wereldbevolking. De overige 95% van de talen worden gesproken door slechts 6% van de wereldbevolking. Zo'n 3900 van die talen heeft minder dan 10.000 spekers. Als de sprekers van die talen hun taal niet kunnen gaan schrijven is de kans groot dat zulke talen uitsterven en de cultuur, waarvan die taal een integraal onderdeel is, verloren gaat. Er zijn verschillende redenen waarom talen (nog) niet geschreven worden. Een belangrijke oorzaak is de orale cultuur waarin informatieoverdracht door middel van vertelde verhalen de norm is en schriftelijke communicatie de uitzondering, of zelfs gewoon niet bestaat. Een van de belangrijkste technische obstakels is de digitale kloof die het schrijven met behulp van een computer voor veel mensen onmogelijk maakt. Dat kan komen doordat er geen (vrije) fonts beschikbaar zijn of doordat het gebruikte schrift dermate complex is dat een 'standaard-oplossing' geen oplossing is. Daarnaast is de beperkte beschikbaarheid of de afwezigheid van lees- en schrijfonderwijs in de moedertaal voor sprekers van veel minderheidstalen een extra barrière.

Taal, schrift en font

Een paar keer is het al gegaan over ‘taal’, ‘schrift’ en ‘font’, maar wat is wat precies en wat is het verschil tussen het een en het ander? Een taal is niet hetzelfde als een schrift en een schrift is niet hetzelfde als een font. Toen mensen eenmaal hun taal gingen schrijven gebruikten ze karakters om de spraakklanken op papier vast te leggen. Deze karakters vormen samen met bijvoorbeeld de leestekens een ‘schrift’ of schriftsoort. Wij gebruiken voor de Nederlandse taal het Latijnse schrift, maar we zouden het ook best in het Cyrillisch of het Arabisch(-e schrift) kunnen schrijven. Op dezelfde manier kan een Arabische (taal) tekst in het Latijnse schrift geschreven worden. De combinatie van een schrift en een taal heet in het jargon een schrift-systeem (writing system). Dit is een combinatie van alle karakters, samen met de regels die beschrijven hoe die karakters gebruikt moeten worden in die specifieke taal. De minderheidstalen van de wereld maken vaak gebruik van het schrift van de meerderheidstaal in hun omgeving. Omdat er in zo’n minderheidstaal bijvoorbeeld andere klanken voorkomen, is het soms nodig om symbolen of accenttekens aan het schrift toe te voegen die in de meerderheidstaal niet nodig zijn. Ook kan het zijn dat bepaalde karakters of tekens op een andere manier gebruikt worden dan in de meerderheidstaal. Om die reden is er vaak per taal een verschillend ‘schrift-systeem’ te beschrijven, zelfs al zijn de (meeste) tekens hetzelfde. Voor zover we weten zijn er minstens zo’n 150 verschillende schriften. Zestig daarvan worden niet meer gebruikt (denk daarbij aan het spijkerschrift of aan Egyptische hiëroglyfen.) Negentig schriften worden op dit moment wel gebruikt. Ongeveer 50 daarvan zijn opgenomen in Unicode, de internationale standaard voor schrift-gebruik, 40 (nog) niet. Een font is een stukje software waarin onder andere de vorm van de verschillende karakters staat beschreven. Verder kunnen fonts informatie bevatten over hoe een karakter in een bepaald schriftsysteem gebruikt moet worden. Met name voor schriften als het Arabisch en de Aziatische schriften zijn deze ‘rendering smarts’ onmisbaar voor de juiste weergave van de gebruikte karakters. Een font kan dus verschillende schriftsystemen ondersteunen. Tenslotte is er nog het onderscheid tussen een karakter en een glyph. Het begrip karakter staat voor een abstract betekenisdragend element, zoals dat bijvoorbeeld in de Unicode tabel is beschreven. Het karakter ‘a’ is ‘Latijn kleine letter a’ in Unicode-termen. Een glyph is de specifieke vorm die de kleine letter ‘a’ krijgt in een bepaald font. Op een abstract niveau is het hetzelfde (een karakter ‘a’) maar op het niveau van de weergave ziet het er verschillend uit. Vergelijk bijvoorbeeld de ‘a’ in het woord meertalig in de titel boven dit artikel met de ‘a’ in het woord ‘talen’ in de ondertitel. Dit zijn verschillende glyphs voor hetzelfde karakter.

Schrijven wereldwijd

In de loop van de tijd zijn steeds meer mensen hun taal gaan schrijven. Er zijn verschillende routes om het fonetische materiaal (de spraakklanken) om te zetten in codes op papier (of in klei, of op papyrus of op boomschors). Ons eigen Romeinse (of Latijnse) schrift is een voorbeeld van een alfabetisch schrift waarbij iedere klank ongeveer een karakter heeft. Met behulp van een paar extra tweeklanken (eu, ui) of accenttekens kunnen we de gesproken taal redelijk goed weergeven. Nu zit er in de manier waarop wij onze taal schrijven een behoorlijke redundantie. “Drm s ht gd mglk m d klnkrs wg t ltn. Zekr ls je mt wt leeshulpn knt werkn”. Arabisch en Hebreeuws zijn voorbeelden van zo’n manier van schrijven. Het gebruik van klinkertekens als een soort leeshulp om de vastgestelde tekst duidelijk te maken laat overigens wel zien dat het toch ook weer niet zo gek is om de klinkers toch op te schrijven.

Voorbeeld 1: arabisch schrift

r.1 ي ب ر ع ل ا

r.2 يبرعلا

Het Arabisch is een schrift waarin de karakters afhankelijk van de plaats in het woord een verschillende vorm hebben. Daarnaast smelten opeenvolgende karakters samen tot zogenaamde ligaturen. Het bovenstaande voorbeeld laat dat zien: regel 1 bevat de afzonderlijke karakters, regel 2 de 'aan elkaar geschreven' vorm.

Overigens is het Arabische schrift ook een voorbeeld van het feit dat schrift niet alleen te maken heeft met techniek, maar ook met cultuur en geschiedenis. Het Arabische schrift kent een lange, rijke traditie van vormen en variatie in de kalligrafie (het met de hand geschreven schrift.) Een van de debatten in de wereld van Arabische fontontwerpers gaat over de vraag of er voor het Arabisch een zelfstandige typografie (door een machine geschreven schrift) mogelijk en wenselijk is. Sommigen menen dat fonts de traditionele kalligrafische vormen en variaties tot in de details moeten volgen. Volgens anderen is er ruimte voor een duidelijke eigen stijl in de fonts die nu ontworpen worden.

Voorbeeld 2 : aziatische schriften
Helemaal aan het andere uiterste van het spectrum zitten benaderingen van schrift die niet uitgaan van de vorm (de klanken) maar van de betekenis (de abstracte concepten), zoals bijvoorbeeld pictogrammen. Schriftsoorten zoals het Japans of Chinees vallen in deze categorie. Weer een andere benadering komt voor in verschillende Zuid-Oost Aziatische talen, zoals bijvoorbeeld het Khmer. Hier is de basis voor het schriftsysteem een medeklinker-klinker-groep. Dit komt ongeveer overeen met de lettergrepen. Er is dus een karakter voor de 'da', voor 'do', voor 'ro', etcetera. Als je met zo'n schrift de klankencombinatie 'door' wilt weergeven schrijf je dus: 'doro'. Voor ons gevoel lijkt het alsof de letters o en r van plaats zijn veranderd, in werkelijkheid zijn ze voor dit schrift op de goede plek geschreven, iedere Khmer-lezer begrijpt dat hier 'door' staat. Hieronder staat nog een voorbeeld van deze 'reordering', nu in het Devanagari schrift.

ब + ि + र + म = बिर्म
b + i + r + ma = birma

Model voor complex schrift

Als we kijken hoe ons eigen Romeinse schrift in de computer verwerkt wordt, dan is er sprake van een redelijk overzichtelijk, eenvoudig model: er is een één-op-één relatie tussen de toetsaanslag en het geheugen en tussen het geheugen en de uitvoer (beeldscherm/printer.) Om de diversiteit aan complexe, niet-Romeinse schriften goed te kunnen verwerken is een ingewikkelder model nodig. Aan de invoerkant moet er vaak gewerkt worden met Input Method Editors. Dit zijn hulpmiddelen die het mogelijk maken om karakters uit een schrift met duizenden karakters te kiezen. Dit kan bijvoorbeeld door middel van een apart invoerscherm of door speciale software die een fonetische transcriptie omzet in de gewenste karakters. In het geheugen worden de gegevens opgeslagen als een verzameling Unicode-karakters. Ook aan de uitvoerkant moet een computer om weten te gaan met speciale eisen die een complex schrift stelt, zoals bijvoorbeeld de volgordeverandering in het Devanagari-voorbeeld hierboven.

Enter XeTeX

XeTeX is een uitbreiding aan de standaard TeX engine die alle systeemfonts eenvoudig toegankelijk maakt voor de gebruikers. XeTeX werkt op Mac OS X, Linux en Windows en zit standaard in TeXLive 2007. Voor de meeste gebruikers is de belangrijkste reden om XeTeX te gebruiken de toegankelijkheid van de systeemfonts. Alle OT, TT en PS fonts die in een willekeurig tekstverwerkingsprogramma te gebruiken zijn, werken ook met XeTeX. Daarbij kunnen de font-setup troubles die zo kenmerkend zijn voor TeX achterwege blijven. Ook kunnen slimme fonts gebruikt worden in combinatie met speciale layout-engines die er voor zorgen dat de gebruikte fonts zich gedragen volgens de regels die gelden voor het betreffende schriftsysteem. Voorbeelden hiervan zijn AAT (Apple's Advanced Typography)[2] op de Mac, of het gebruik van de ICU-library[3] en SIL's Graphite techniek[4]. XeTeX is in de TeXwereld enthousiast ontvangen. In LaTeX zijn er verschillende packages gemaakt die XeTeX ondersteunen, waarvan het fontspec-package wel de belangrijkste is. Verder herkennen veel gebruikte packages zoals graphics, hyperref en PStricks het gebruik van XeTeX automatisch. Ook ConTeXt kan prima met XeTeX overweg[5]. Waar luaTeX met behulp van de ingebouwde scriptingtaal Lua de TeX-engine toegankelijk maakt voor bijvoorbeeld OT support, gaat XeTeX een andere weg: het maakt gebruik van libraries op besturingssysteemniveau zoals fontconfig (voor het vinden van de fonts) en de al genoemde AAT, ICU en Graphite.

Voorbeeld 3 : Typografische features in OT fonts
Hier volgen een paar voorbeelden van hoe XeTeX typografische features van OpenType fonts toegankelijk maakt. De tekst "Hallo Wereld! 0123456789" kunnen we in het font Garamond Premiere Pro zetten (voorbeeld 1). Met de aanduiding +smcp worden de letters als klein kapitalen gezet (voorbeeld 2) en met de code +sups selecteren we de optie superscript, wat voor dit font blijkbaar niet werkt voor hoofdletters en leestekens.

standaard:"GaramondPremrPro" Hallo Wereld! 0123456789
var1: "GaramondPremrPro:+smcp" HALLO WERELD! 0123456789
var2: "GaramondPremrPro:+sups" H$^{\text{allo}}$ W$^{\text{ereld}}$! ^{0123456789}

Een ander voorbeeld van stylistische variaties, deze keer met Apple's AAT techniek, laat het volgende set met variaties van het Apple Chancery font zien.

"Apple Chancery:Design Complexity=Simple Design Level" at 16pt.
"APPLE CHANCERY:LETTER CASE=SMALL CAPS" AT 16PT.
"Apple Chancery:Design Complexity=Flourishes Set A" at 16pt.
"Apple Chancery:Design Complexity=Flourishes Set B" at 16pt.
"Apple Chancery:Design Complexity=Flourishes Set C" at 16pt.

Voorbeeld 4 : Taal en schrift features in XƎTEX
Met de Unicode ondersteuning kan XƎTEX het ideaal van ‘iedere taal en ieder schrift’ ook voor TEX-gebruikers een stapje dichterbij brengen. Hier volgen een paar voorbeelden van taal- en schrift-specifieke features.

Speciale karakters zoals de Bengaalse klinker O worden geschreven met twee los van elkaar staande tekens die samen de medeklinker insluiten. Het voorbeeld hieronder geeft eerst de afzonderlijke karakters zonder speciale weergave, daarna de juiste, samengevoegde weergave die te activeren is met de code: ’script=beng’.

zonder 'script' aanduiding

<LETTER KA> : ক <VOWEL O> : ো → কো

---

met 'script=beng'

<LETTER KA> : ক <VOWEL O> : ◌ো → কো

Het font Charis SIL[6] is een voorbeeld van een font met speciale taalspecifieke coderingen. In het onderstaande voorbeeld van een tekst in het Vietnamees is het verschil in de plaatsing van de accenttekens te zien. Deze speciale plaatsing is van belang voor de juiste weergave van het Vietnamees.

Unicode cung cấp một con số duy nhất cho mỗi
Unicode cung cấp một con số duy nhất cho mỗi

In het kader van dit korte artikel is het onmogelijk om een overzicht te geven van alle verschillende soorten schriften en manieren om daar TEXnisch mee om te gaan. Hopelijk maken deze paar voorbeelden in elk geval duidelijk dat er op dit gebied veel mogelijk is.

Noten

1. Raymond G. Gordon, ed., Ethnologue: Languages of the World, 15th edition, 2005. Zie voor de digitale versie: www.ethnologue.com
2. Zie ook: developer.apple.com/textfonts/
3. Zie ook: www.icu-project.org/
4. Zie ook: scripts.sil.org/RenderingGraphite
5. In elk geval meestal, zie ook: wiki.contextgarden.net/XeTeX
6. Zie ook: scripts.sil.org/CharisSILfont

Jelle Huisman
SIL International
`jelle_huisman (at) sil (dot) org`

What is it about all those *TEXs

Abstract
This short article describes the different 'layers' in a TEX system, the differences between TEX engines, extensions, macro packages, and distributions. I hope to take away some of the confusions that people new to TEX and less technically inclined people have when they are confronted with terms like `pdftex`, `texlive`, `tetex`, `miktex`, `pdflatex` and so on.

Keywords
TEX, LaTEX, miktex, tetex, TEXLive, etex, pdftex, pdfetex

Introduction

When you want to use TEX on your computer you are quite soon confronted with a lot of software parts which have names that end on 'tex'. If you are lucky you have a computer which has TEX pre-installed. Otherwise you will have to choose what to install. You might get a DVD with one install-button that installs a complete system for you, but more often you will have to choose which components you want to install. And if your TEX installation has become outdated you may come to the point that you have to install a new system. Sometimes it is just an upgrade of the existing installation but there comes a point when that is no longer supported and then you have to decide what to choose. Now this choice can be quite confusing. Do you have to use 'MiKTEX' or 'TEXLive', 'TeXShop' or 'TeXnicCenter? And what are these things? Why doesn't installing TeXShop on a Mac give you a working TEX system? And if you want to process your files do you choose 'latex' or 'miktex'? We will see that the last question is the wrong one but for new users it seems a logical thing to ask. Below I will try to put some order into all these terms so that you will be able to place the different things in their proper place and to know which questions have meaning.

The beginning

In the beginning there was TEX. This is the original program designed and programmed by Donald Knuth. You prepared your input according to the specifications in The TEX Book. This style of TEX documents is called 'plain TEX'. You would run `tex` from the command line (there were no graphical user interfaces in that time) on this input and a DVI file would be produced. There was a program to preview the DVI file and one or more programs to send it to a suitable printer. And there was a companion program `metafont` but that was only used by some font freaks. And that was it.

Further developments

After the original TEX implementation several new developments have occurred.

- □ *Different syntax forms for your input document.* Plain TEX is quite simple but out of the box it doesn't support more advanced features, like cross references, automatically numbered sections and so on. You can program these yourself but it is tedious and it makes exchange of documents with other authors more difficult. It would be nice if you could use the work that others have done. Such collections of definitions and pre-programmed features are called 'macro packages'. The most well known of these are LaTEX and ConTEXt. Please note that these macros do not require changes to the TEX program but are a kind of additional input files for TEX, extending the input language TEX understands with new – usually higher-level – commands.
- □ *Extensions of the TEX program.* There are several limitations in the TEX program that make it hard for writers of macro packages to program some advanced features. Another wish people had is to produce PDF files directly rather than, or better, in addition to, the arcane DVI format. Or to make easier use of the fonts that are present in modern operationg systems. For a good programmer it is not very difficult to enhance the TEX program to overcome these difficulties. However, Knuth does want TEX to be stable, so he doesn't change it anymore. He allows others to make changed versions, however, as long as they are not called TEX. Therefore all kinds of extensions of TEX have appeared under names such as eTEX (additional facilities for macro programmers), pdfTEX (PDF output as an option), or XeTEX (support for

operating systems fonts).

- □ *Engines*. Most of the macro packages can be combined with most of the (extended) programs mentioned above. Often the combinations are prepackaged in such a way that they can be executed as a specific program. We call such a program an 'engine'. (Some people might use the word 'engine' only for the above mentioned programs without reference to a preloaded macro package.) An engine can be for example `latex`, `pdftex` or `pdflatex`.
- □ *Additional programs*. Having only TEX and `metafont` wasn't sufficient. Many other supporting programs have been added, such as programs to manipulate bibliographies, picture processing, support of different languages, and so on. Well known ones are `bibtex` for bibliography processing and `makeindex` for the sorting and formatting of alphabetical indices.
- □ *Graphical user interfaces* to make working with TEX and the additional programs easier. Nowadays most computer users do not feel comfortable with command line interfaces. Therefore there are graphical programs, sometimes called IDE's (integrated development environments). With these you can edit your input documents, often with syntax coloring. And they contain menus and buttons for seamlessly activating the major programs that are part of a TEX processing cycle, including viewing errors, previewing the document and printing it. Examples are TeXShop, TeXnicCenter, and Texmaker
- □ *Many other additions* are useful, for example support for many fonts, additional macro packages as additions to LaTEX, ConTEXt and others, and so on. Together such a TEX system easily contains tens of thousands of files. Managing these files and keeping up with new releases of them and keeping them consistent is a major job. Therefore there are *distributions* which consist of a carefully selected and tested collection of files and programs that together form a working environment for the processing of TEX documents.

In the rest of this article I will deal with the above mentioned parts in a 'top-down' manner, i.e., starting with the things the user will encounter first, and then digging down to a more detailed view of how these things are built up internally. As we go into more details some of the above information will be repeated below.

Distributions

A distribution is a complete collection of TEX-related programs of all thinkable sorts, macro packages, documentation files, supporting programs, etc. They usually come on a CDROM or DVD (most don't fit on a CDROM anymore these days) or can be downloaded from the Internet.

If you install a distribution you have in principle a working and very complete TEX system, but usually not a GUI. The advantage of installing a complete distribution rather than collecting the parts yourself is that it most probably has been thoroughly checked and all the parts have versions that are supposed to work together.

Popular distributions are:

- □ MiKTEX. Built and distributed by Christian Schenk for MS-Windows. Some parts of it have been ported to GNU/Linux and Mac OS X. This is probably the most popular distribution for MS-Windows. The great advantage of MiKTEX is that it has an advanced 'package manager'. This is a part that automatically installs missing parts if they are needed. So you can choose to install a rather small subset of MiKTEX initially, and let the package manager automatically download and install additional needed parts. Of course you must be connected to the Internet for this to happen. The package manager can also be used to upgrade outdated parts or install new parts manually.
- □ teTEX. This used to be a large distribution for Unix-like systems packaged by Thomas Esser. Many GNU/Linux systems still have this as their default distribution. However, it is no longer maintained. It has been replaced by:
- □ TEXLive. This distribution was originally based on teTEX but the choice of what it included was different from teTEX. It also supports MS-Windows, GNU/Linux and Mac OS X, and is therefore a good choice when portability is important. It is released on a CDROM/DVD set and freely distributed (about yearly) to the members of most TEX user groups, like NTG. It can also be downloaded from the Internet.
- □ MacTEX is a distribution for Mac OS X which is a subset of TEXLive that can be downloaded from the Internet.

GUI's

A GUI (Graphical User Interface) is a program or collection of programs to make your life as a TEX user easier. They contain an editor for entering and modifying the TEX input file(s), often with syntax highlighting. And they usually have buttons to run the proper engine on the file, display it on your screen, print it, and run supporting programs like bibliography and index processing.

They usually don't include the TEX programs and macro packages but suppose that you have installed these from a distribution. So for example installing TeXShop in your Mac doesn't help you very much if you don't also install a complete TEX system like TEXLive or MacTEX.

Popular GUI's are TeXnicCenter and WinShell on MS-Windows, TeXShop on Mac OS X and TeXmaker for GNU/Linux as well as MS-Windows and Mac OS X. On Unix and GNU/Linux systems many people use the Emacs editor as a GUI for TEX.

Macro packages

We have already mentioned the macro packages `plain`, LaTEX and ConTEXt. A macro package consists of a collection of command definitions and some such, and are contained in one of more text files. For efficiency reasons they are preprocessed and stored in a binary file called a 'format file' or '`fmt`' file. A TEX program (in the extended sense) can easily load a format file, either by specifying it as a command parameter or by 'cloning' the engine program file and giving it a name corresponding to the name of the format file. So for example the program `latex` was the TEX engine with the LaTEX format preloaded, and `pdflatex` is the pdfTEX engine preloaded with the LaTEX format.

Even if you have written your own macro package it is easy to generate preloaded engines and give them a required program file name.

XeTEX can also be loaded with a LaTEX format and the resulting program is called `xelatex`.

Engines

The TEX program that translates your input file to a DVI or PDF file we have called the TEX engine. The program file that contains the original engine is usually just called `tex` (or `tex.exe` on MS-Windows systems). The TEX engine doesn't change very much as Knuth has decided that he will not make any more enhancements to it despite the limitation the program has. And as it is mature bugfixes are also rare.

Other people, however wanted to add new features. This is allowed by the license, but the resulting program must not be called 'TEX'. Such a program is also an engine, but then not for TEX but for an extension of TEX. Strictly speaking they should therefore not be called TEX-engines, although in sloppy speaking this is often done.

The most important extensions of TEX are (or were):

- eTEX. This extension of TEX adds typesetting for languages that are written right-to-left, like Hebrew and Arabic. Left-to-right and right-to-left typesetting can be intermixed. It also expands the number of 'registers' that TEX has for storing numbers, dimensions and so, which, with complicated packages sometimes were exhausted. There are also some other extensions. LaTEX and ConTEXt nowadays are dependent on some of these extensions.
- pdfTEX. This is an extension that adds direct PDF output besides DVI output. It was developed in a PHD research by Hàn Thế Thành. It also contains microtypographics extensions which were the main subject of his thesis.
- pdfeTEX. This used to be a combination of eTEX and pdfTEX, but now pdfTEX contains the eTEX extensions itself thereby obviating a separate pdfeTEX engine. And as this contains all the possibilities of the original TEX and eTEX, it has become the default engine for most TEX applications. Only plain TEX uses the original engine from Knuth.
- XeTEX adds the possibility of using your operating system's fonts in a simple way, and it supports Unicode input and the use of OpenType fonts. It incorporates the eTEX extensions and some of the pdfTEX extensions. It has a dvi backend but works closely with dvipdfmx to produce PDF. The program file is usually called `xetex`.
- luaTEX. This is a relatively new extension, still in development, which adds the Lua programming language to the pdfTEX engine. This gives an enormous flexibility. It also supports OpenType fonts. It is meant to become the successor of pdfTEX. The program file is called `luatex`.
- Others. There are other engines, such as Omega and Aleph but they are experimental and often buggy and therefore not much used. Aleph functionality is being merged into `luatex`.

The engines are the first layer on which a TEX system is built. Most people use a preloaded engine, however, which is a combination of a TEX (extension) program and a macro package. We will take an engine preloaded with LaTEX as an example.

Nowadays always the pdfTEX engine is used (as described in the previous section including the eTEX extensions). Both the program `latex` and the program `pdflatex` are engines consisting of `pdftex` preloaded with the LaTEX macro package. The difference is that the program `latex` is parametrised to generate DVI output and `pdflatex` to generate PDF output. But note that both can also produce the other output if the document source would choose so.

The program `context` is the `pdftex` engine preloaded with the ConTEXt format, but it can also use other engines, such as XeTEX. ConTEXt comes in different language flavors, like `cont-en` for the English version and `cont-nl` for the Dutch version. `context` is usually not called directly, however, but by means of the program `texexec`.

Summary

engines:
`tex, pdftex, xetex, luatex`

macro packages:
Plain, LaTEX, ConTEXt

engines with preloaded macro packages:
`tex, latex, pdflatex, xelatex, context`

GUI's:
TeXnicCenter, WinShell, TeXShop, Texmaker

distributions:
MiKTEX, teTEX (obsolete), TEXLive, MacTEX.

I would appreciate to get feedback to be able to improve on this overview. If you have additions or feel that some things are unclear please email me at the address below.

Piet van Oostrum
piet@vanoostrum.org

Book Review: Fonts & Encodings *by Yannis Haralambous*

Keywords
fonts, encodings, book review

Yannis Haralambous is well known in the international TEX community, not only as a co-founder of the Omega project, but also for his numerous contributions as a developer of fonts for various languages. It only seems fitting that Yannis has undertaken the job of writing a comprehensive book on the topic of fonts and encodings.

The original edition, entitled *Fontes & Codages*, appeared in 2004, but only in French. Now, a long-awaited English translation as *Fonts & Encodings*, prepared by P. Scott Horne, has recently become available published by O'Reilly Media Inc.

Contents

Amounting to a little bit more than 1000 pages the book matches the size of *The LaTEX Companion, 2nd edition*. It appears quite impressive, not only regarding its sheer size, but also regarding the broad range of topics covered as well as the depth of the coverage and the level of detail. In some cases the author has spent dozens of pages documenting some arcane details of font formats, which have so far been lacking a comprehensive or accessible documentation from other sources.

The book consists of a main body of 14 chapters of some 600 pages, followed by an appendix of 7 chapters amounting to another 400 pages.

As the author makes clear in the introduction, various groups of readers may benefit from different parts of the book without having to read all of it: Some chapters are mostly encoding-specific, dealing with characters on the input side, some are mostly font-specific dealing with glyphs on the output side, while other chapters find themselves in the middle ground, having to deal with font and encoding topics simultaneously. Some chapters are accessible to end users interested in installing and using fonts, while others are only of interest to font designers or developers of font-related software.

The first part of the book starts with encoding-specific topics. Chapter 1 provides an overview of the history of encodings before Unicode, ranging from 7-bit ASCII and various 8-bit ISO encodings to 16-bit East-Asian encodings. Chapters 2–4 cover the Unicode standard, starting with an overview of the symbols and scripts included in the standard and moving on towards more and more complex implementation details. Chapter 5 completes this part with a presentation of some useful tools for using Unicode input on various system platforms.

The second part covers the topic of font management on various system platforms and operates somewhere in the gray area between fonts and encodings. Chapters 6–8 each cover similar topics for the Macintosh, Windows and Unix/X11 platforms. While the description of the Macintosh platform is rather detailed in discussing the differences of font handling between MacOS 9 and MacOS X, the description of the Unix/X11 platform only covers some very basic and old-fashioned X11 tools. Here, one could have wished for some more extensive coverage of font management in modern Linux desktop environments such as KDE or Gnome.

The following two chapters discuss platform-independent usage of fonts in TEX/Omega systems and on the Web. Chapter 9 starts with an overview of high-level font selection in LaTEX/NFSS2, followed by a detailed description of low-level font installation for dvips. The remainder of the chapter then discusses numerous examples of creating virtual fonts using fontinst to implement specific effects needed in various scripts. Chapter 10 concludes this part with a coverage of fonts on the Web using either (X)HTML/CSS or alternatively SVG.

The final part of the book covers font-specific topics. Chapter 11 covers various classifications of of Latin typefaces and simultaneously provides a nicely-illustrated overview of the history of the most important typeface designs. Chapters 12–13 than discuss creating, editing and optimizing PostScript, TrueType, and OpenType fonts using tools such as FontLab or FontForge. Chapter 14 finally introduces the concepts of advanced typographic features provided in OpenType or AAT fonts and discusses ways of enriching fonts using these facilities.

The appendix of the book mostly consists of the detailed descriptions of font formats. Starting from bitmap fonts and TEX-related font formats and moving on to PostScript, TrueType, OpenType and AAT fonts, practically all relevant font formats are covered in detail in Appendix A–E.

Finally, Appendix F discusses the principles of font design in MF and derived systems such as MP, MetaFog and MetaType 1.

Commentary

Considering the size of the book, it is understandable that several years have passed from the time of writing the manuscript to the publication of the English translation. Unfortunately, because of this, some chapters of the book are in risk of becoming out-of-date rather quickly. For most of the material, we have to assume that the English edition of 2007 only represents the state of the art of 2003.

For many chapters serving as a reference, such a delay is not much of a problem, as the descriptions of encodings or font file formats remain unchanged and permanently valid. On the other hand, however, it is regrettable that especially the chapter about TEX has completely missed or overlooked some very important developments of the last few years.

As one example, when describing the details of font installation, the author only covers TEX/Omega with dvips, while PDFTEX isn't mentioned in this context, even though most of the description would be applicable to both systems in TEX Live systems. (In fact, PDFTEX isn't mentioned anywhere at all in the book, perhaps because it didn't support Omega at that time and was therefore seen as line of development that was irrelevant to the author.)

As another example, Hàn Thế Thành, author of PDFTEX is only mentioned once in the context of Vietnamese fonts, while his significant achievements regarding the implementation of micro-typographic features of PDFTEX have been neglected completely. This is even more surprising as the author spends some pages discussing an example where the effect of *margin kerning*, which would have been accessible in PDFTEX, is simulated in a rather cumbersome way using virtual fonts created by fontinst.

Regarding examples of extensions of Computer Modern fonts, the author suggests the CM-Super fonts, while the (by now) much more popular Latin Modern fonts are only mentioned in passing as an example of a MetaType 1 application.

Finally, when it comes to discussing ways of using advanced typographic features of OpenType or AAT fonts in TEX, the author only offers some hints about his own research work in the Omega 2 project (which hasn't progressed beyond the prototype stage), while the (by now) readily-available newcomer XƎTEX remains unmentioned.

To be fair, one has to admit that the success and importance of these recent developments in the TEX world could not have been foreseen at the time of writing in 2003. Nevertheless, it could have been possible to include some additions and/or revisions by the time of the English translation of 2007.

It is rather unfortunate that the opportunity for updates was missed here, which would have made the book much more useful and valuable for TEX users interested in making use of the very latest of developments in font technology.

Despite these shortcomings, the book remains a valuable resource for TEX users and software developers, who are deeply interested in font technology and encodings. There is no other book providing a similar coverage in the broad range of topics and the deep level of detail in a single volume. To get anywhere near it, one would have to collect dozens of references from a variety sources and one would still be left with some gaps to fill.

In summary, this book is certainly recommendable. Nevertheless, some additions and/or revisions would be very desirable for future editions.

As as final remark, the reviewer would like to mention a little curiosity: Like some other modern textbooks, this book also features a separate index of persons besides the usual general-purpose index. However, unlike other books, the author seems to have been rather liberal as to which kinds of persons are referenced in the index.

This way, authors of font software and tools not only find themselves in the glorious company of some the most famous font designers of history, but also in the vicinity of rather questionable political figures (e. g. Lenin, Hitler, Mao) as well as some fictional or literary characters (e. g. Sherlock Holmes, James Bond, James T. Kirk), which are only mentioned in passing in some light-hearted comments.

While the reviewer (who at one time was also contributor of fontinst) wishes to express thanks for the opportunity of being included in this rather unique selection of people, he also wishes to express serious doubts, whether it is really helpful to the reader to include fictional characters in the person index in the same way as technical people. This practice is certainly debatable and probably should be reconsidered for future works of this kind.

Yannis Haralambous, translated by P. Scott Horne, Fonts & Encodings, O'Reilly Media, 1016+xx pp., first edition: September 2007, ISBN*: 0-596-10242-5, US$ 59.99*

Ulrik Vieth

On reading Fonts & Encodings

Abstract
Stated briefly: "Should I buy this book ?"

YES.

In fact I suppose you are reading this article because you are in a someway involved with TEX, or LaTEX, or ConTEXt. Maybe for fun, or for daily works, like me (well, I am bit lucky, for me it is fun *and* work).

If so, then you cannot disagree with these simple facts:

1. Unicode is becoming *the* standard for electronic text interchange, but others/old encodings are still present due to legacy systems;
2. today, and more than some years ago, one can choose his/her Operating System (OS), and, without make any tort to anyone, today we have Macintosh, Windows and Linux, a quite diffused playing field.
3. today we are using the WEB, or WEB-Applications (i.e. spreadsheets online, rich text editor online), not only to see correct texts but also to *cut-and-paste* text from the WEB into our own texts.

So we quite often must match (or fight with) text-OS-WEB, and occasionally we jump onto the encoding-font-mess (What is the encoding of this text? Do I have the right glyphs?).

This book covers these subjects in the ten chapters (from 1 to 10, but chapter 9 is dedicated to fonts in TEX which anyway fits well in between) and, of course, one can also find here the relations between Unicode and typography, especially in chapter 3 and chapter 4, a prelude to the rest of the book.

Even if useful, these first ten chapters are only an exercise, and maybe a bit difficult.

So, why not take a breath with something more relaxing and read a chapter about "The History and Classifications of Latin Typefaces" (chapter 11)?

And, after viewing some glyphs, maybe you want to draw something on your own, and hence go on with chapter 12, and chapter 13 and chapter 14, and maybe you will find that "Fonts, Encodings & Tools" is not really an inappropriate title.

Well, and now? What's next?

Nothing.

The book is over.

Really.

So if you want to stop here, no problem. You have read almost half a book, but it is complete.

If you do go on, be warned: it is technical material, as it must be. In the following 7 appendices (almost 600 pages, a bit less than half of total pages) you will read about fonts from bitmap to PostScript to TrueType to OpenType to MetaFont ending with Bézier curve, with no compromises to readability: editing/creating fonts is not for infants, expecially today.

But, having said that, I must admit that it is also a pleasure to read these appendices; and, in the end, the final surprise is a bibliography with 358 entries.

Of course, one cannot ignore that sometimes some notes are. . .well, funny, and some non-tecnichal opinions may be not so condivisible as they seems; also it seems that the book talks about a Fontforge from 2004 (a bit outdated), and, well, TTX is not so robust (I have found some OpenType fonts with OS/2 = 3 that break TTX). Instead, one should download last source from http://fonttools.sourceforge.net/cvs-snapshots/zip which appears to be more stable.

And nothing at all is said about pdfTEX or luaTEX (but there is a wiki, see http://luatex.bluwiki.com/go/Luatex).

But even so, I enjoyed reading this book.

Thank you very much, Mr. Yannis Haralambous.

Luigi Scarso

Latin Modern Nederlands

De Latin Modern-fonts zijn inmiddels geen onbekende meer en zijn al aardig geïntegreerd in de TEX-distributies. Deze fonts zijn de uitkomst van het Latin Modern-project dat door de TEX gebruikersgroepen, waaronder de NTG, wordt gefinancierd. De doelen van dit project laten zich als volgt samenvatten:

- het combineren van de tot dan toe gebruikte Type1-fonts in slechts een font (voor pdfTEX)
- het voorzien in de behoefte aan een OpenType font (voor XƎTEX en LuaTEX)
- het opschonen van de shapes en corrigeren van bekende bugs
- het aanvullen van het repertoire aan Latin glyphs

De fontset vervangt in TEX-distributies de traditionele Computer Modern-fonts en biedt daarbij wat extra monospaced varianten. Over de jaren heen waren nogal wat aanvullingen ontstaan op CM en geen ervan was compleet zodat schrijvers van macropakketten zich nog wel eens van rommelige constructies moesten bedienen om de gewenste karakters in de uitvoer te krijgen.

Aansluitend is het TEXGyre-project gestart, dat dezelfde doelen verwezenlijkt voor tien veel gebruikte fonts in de TEX-distributies. In de wandelgangen wordt daarbij vaak gesproken over het 'lm-ineren' van fonts, wat dan zowel duidt op het synchroniseren van de beschikbare glyphs als het verminderen van het aantal fontfiles. Wat geldt voor LM, dat geldt ook voor Gyre.

Het verder opwerken en completeren van de standaard LM-fonts resulteert in regelmatige updates van de resulterende files. Eind januari is versie 1.05 uitgebracht. In deze versie zijn weer wat extra glyphs beschikbaar (vooral ten behoeve van transliteraties) maar de meest drastische wijzigingen zitten in de OpenType varianten:

- nieuwe filenamen zodat in andere applicaties dan TEX-gebruik handiger is
- een meer robuuste ondersteuning van features zoals ligaturen
- een andere systematiek van interne naamgeving

Er is door een groep betrokkenen zo goed en kwaad als dat kan gekeken in hoeverre de verschillende besturingssystemen en grafische applicaties fonts presenteren en de huidige interne naamgeving is het best mogelijke compromis. Natuurlijk kunnen TEX-macropaketten er hun eigen laag overheen leggen en veel voor de gebruiker verbergen.

Er is verrassend veel tijd gaan zitten in een voor de Nederlandstaligen specifiek probleem: de ij. Om een of andere reden vinden we het vanzelfsprekend als we in een font een fi, fl, ffl en ffi ligatuur aantreffen. Overigens, zulke ligaturen zijn vooral bedoeld om dichtlopen door inkt te voorkomen en niet zozeer esthetisch. Als gevolg van meer moderne druktechnieken vinden we in hedendaagse fonts minder vaak zulke ligaturen. Echter, wat te denken van onze zogenaamde 'lange ij'. Op school leren we deze combinatie van een i en een j als een karakter te schijven en het zal dan ook niemand verbazen dat er in Unicode twee slots zijn gereserveerd: `0x0132` voor IJ en `0x0133` voor ij. Zowel in Unicode als in de glyphname worden dit ligaturen genoemd, maar de meeste Nederlandstaligen zullen het als een letter zien. Het is de vraag of bij het opdreunen van het alfabet op school kinderen denken aan deze ij of aan de zogenaamde griekse y.

Hoe dan ook, met de komst van OpenType zullen we de ligatuur vaker zien verschijnen. De reden hiervoor is niet dat plotseling iedereen het gereserveerde Unicode slot gaat gebruiken (we blijven gewoon i+j intypen), maar dat fonts ingebouwde regels hebben voor het vervangen van combinaties van karakters. Dit is niet nieuw, denk maar aan de fi ligatuur die opduikt als we een f en een i intypen. Echter, waar deze ligatuur in vrijwel alle talen opduikt, is de ij ligatuur alleen in het Nederlands van belang. Dit betekent dat in bijvoorbeeld Latin Modern, waar we een ij-ligatuur hebben, de OpenType-fonts extra slim moeten zijn als het onze taal betreft. Vergelijk de voorbeelden in figuur 1 maar eens.

In een OpenType font zijn features gekoppeld aan een script (zoals latin) en een taal (zoals Nederlands). Beiden hebben standaard de waarde 'default'. Een van de beslissingen die een fontontwerper (of degene die het font technisch afhandelt) moet nemen is waar precies zo'n ligatuur moet worden ondergebracht: is het een 'required' ligature (rlig), een 'discretionary' ligature (dlig), een gewone ligatuur (liga) of misschien een 'historic' (hlig)? De praktijk leert dat fonts op dit vlak niet consistent zijn, iets wat niet verwonderlijk is, omdat de OpenType-standaard minder open is dan de naam suggereert. We zullen er mee moeten leren leven dat weliswaar veel font-gerelateerde zaken (om er een

script	taal	liga feature	resultaat
dflt	dflt	disabled	fijn ijs eten in de file
dflt	dflt	enabled	ﬁjn ijs eten in de ﬁle
latn	dflt	enabled	ﬁjn ijs eten in de ﬁle
latn	nld	enabled	ﬁjn ijs eten in de ﬁle
latn	eng	enabled	ﬁjn ijs eten in de ﬁle

Figuur 1.

te noemen: installatie) eenvoudiger worden maar dat de eindgebruiker nog steeds voor beslissingen wordt gesteld.

In Latin Modern is onze ij beschikbaar als ligatuur waarvan de i en de j wat dichter bij elkaar staan dan normaal. Hier zou dus ook taal- en context-afhankelijke kerning kunnen worden gebruikt.

Wat is dat, een context-afhankelijkheid? Ook in het geval van een ligatuur moeten we bij vervanging van de i en j door een ij kijken naar meer dan alleen deze twee letters. Immers, wat verwacht men in het woord fijn? Een fi+j+n, of een f+ij+n. In Latin Modern zijn regels opgenomen die inhouden dat in geval van een f gevolgd door een i, gevolg door een j eerst de laatste twee karakters worden vervangen door een ij. Vervolgens hebben we een f gevolgd door een ij en dat is geen paar dat tot een ligatuur leidt. Dit soort truukjes kan alleen door XƎTEX en LuaTEX worden uitgevoerd, op basis van de in het font opgeslagen informatie. Let wel: op zich zou je dit soort regels kunnen vastleggen onafhankelijk van het font (en dus ook kunnen toepassen bij gebruik van Type1-fonts) maar in dit geval liggen de regels vast in het OpenType font.

Gezien het aantal talen en gezien de taalspecieke font gerelateerde (al dan niet historische) zaken, kan men zich voorstellen dat fonts en fontontwerpen er niet eenvoudiger op worden. Ook kunnen we uitzien naar bugs in fonts, incomplete feature-specificaties, beperkte omvang van het aantal regels, enzovoort. Het is verder de vraag of en wanneer we gaan afwijken van zulke regels, immers, TEX-gebruikers hebben vaak zo hun eigen idee over wat mooi en wenselijk is. Hoe dan ook, het blijft opletten geblazen.

Hans Hagen

Theorems in ConTEXt

Abstract
This article explains some of the recent advancements in ConTEXt enumeration mechanism that handles most of the requirements of theorem-like constructions.

Keywords
ConTEXt, theorems, enumerations

Introduction

In mathematics writing important results are usually presented in such a way that they stand out from the surrounding text and can be referenced later on. Theorems, lemmas, corollaries, and proofs are most common examples. In ConTEXt, enumerations can be used to define such theorem-like environments and this article explains how to do that.

The basics

A typical theorem looks like this.

Theorem 1 (Pythagoras Theorem) *The square on the hypotenuse is equal to the sum of the squares on the other two sides.*

It consists of a name (theorem), a number (in this case 1), an optional title (Pythagoras Theorem), and a body. We want to be able to configure the blank space before and after the theorem, the style of the name, number, title, and the body. Some attributes are shared amongst all theorem-like environments, others are specific to a particular environment. All common features can be set up using `\setupenumerations`. For example, in this document I have set

```
\setupenumerations
  [ before={\blank[big]},
    after={\blank[big]},
    location=serried,
    width=broad,
    distance=0.5em,
    headstyle=bold,
    titlestyle=bold,
    way=bytext,
    conversion=numbers]
```

The `before` and `after` keys are for setting up what goes before and after the enumerations. In this example, we use it to set up the blank spaces; later on I will show an example to use these keys to do fancy stuff. The `location` key sets up where the head and the title will be located. There are various options which are documented in the ConTEXt manuals. I use `serried` with `width` set to `broad` and `distance` equal to `0.5em`. This gives the typical style in which theorems are typeset. The keys `headstyle` and `titlestyle` set the style for head (theorem name and number) and title. There are also `headcolor` and `titlecolor` to set up colors. If you really want to fine tune the appearances of head and title, you can use the `headcommand` and `titlecommand` keys to use a custom command. The `way` key configures the numbering; we can use `bytext`, `bychapter`, `bysection`, `bypage`, etc. The `conversion` key sets up the numbering; we can use `Characters`, `numbers`, `Roman`, or a customized conversion.

Once we have set up the style common for all enumerations, we can set up individual theorem-like environments.

```
\defineenumeration
  [theorem]
  [    text=Theorem,
      title=yes,
      style=italic,
       list=all,
   listtext={Theorem }]
```

This defines an environment (`\starttheorem ...\stoptheorem`) with the name “Theorem”, an optional title and italic body style. It also sets a list `all` to store theorems, with the `listtext` as “Theorem ” (notice the space). This enables us to place a list of theorems later on. This environment can be used as

```
\starttheorem[thm:pythagoras]{Pythagoras Theorem}
  The square on the hypotenuse is equal to
  the sum of the squares on the other two sides.
\stoptheorem
```

The `\starttheorem` command takes two optional arguments. The first optional argument is in square brackets and sets up the key for referencing: we can use `\in[thm:pythagoras]` to get the number of the theorem. The second argument is in curly brackets and sets the title of the theorem; the title is typeset according to `titlestyle` and `titlecolor` keys and is surrounded by `titleleft` and `titleright`, which by default are set to (and). If we want a theorem without a title, we can leave out the second optional argument. For example, to get this

Theorem 2 *The square on the hypotenuse is equal to the sum of the squares on the other two sides.*

we type

```
\starttheorem
  The square on the hypotenuse is equal to the sum of the squares
  on the other two sides.
\stoptheorem
```

We can also set up the proof environment using enumerations. A typical proof looks like

Proof I have discovered a truly marvelous proof of this, which this margin is too narrow to contain. $\square$

Normally proofs do not have a number or a title, and have a symbol $\square$ at the end to indicate the end of the proof. Proofs can be set up as

```
\defineenumeration
  [proof]
  [       text=Proof,
        number=no,
     headstyle=italic,
         title=no, %this is the default
   closesymbol={\mathematics{\square}},
         style=normal]
```

The number=no makes the proofs not numbered, the closesymbol key sets the close symbol (□) which is placed at the end of the environment and is pushed to the right edge of the line. The current algorithm for placing the close symbol is fairly simple and does not give the correct result in all cases. Sometimes (e.g., when the proof ends with an itemize environment) one has to manually tell ConTEXt where to place the close symbol by putting a \placeclosesymbol command at the appropriate place. This command is equivalent to the \qedhere command of the amsthm package in LaTEX. The closesymbolcommand key can be used to set a customized command to place the close symbol. The above proof was keyed in as

```
\startproof
  I have discovered a truly marvelous proof of this, which this margin
  is too narrow to contain.
\stopproof
```

Sharing numbers

Suppose we want to define a corollary environment, which shares the same number as theorem. So, if we key in

```
\startcorollary
  The sum of angles of a quadrilateral is $360^{\circ}$.
\stopcorollary
```

we will get

Corollary 3 The sum of angles of a quadrilateral is 360°.

In order to share the numbering with theorems, we need to define corollary as

```
\defineenumeration
  [corollary]
  [    text=Corollary,
     number=theorem,
       list=all,
   listtext={Corollary }]
```

The number=theorem key tells ConTEXt to use the same number for theorems and corollaries. Notice that the number key is smart: number=no means that the enumeration will not be numbered, number=theorem means that the enumeration will have the same number as theorems.

Framed and shaded theorems

In LaTEX, ntheorem package provides a means to get shaded and framed theorems. In ConTEXt, the enumerations provide enough hooks to make this possible. Suppose we want all axioms to be framed with a random frame. First let us define the fancy text background

```
\definetextbackground
    [axiomframe]
    [           mp=background:random,
          location=paragraph,
     rulethickness=1pt,
             width=broad,
        leftoffset=1em,
       rightoffset=1em,
            before={\testpage[3]\blank[3*big]},
             after={\blank[3*big]}
    ]
```

where `background:random` is defined as

```
\startuseMPgraphic{background:random}
   path p;
   for i = 1 upto nofmultipars :
    p = (multipars[i]
     topenlarged 10pt
     bottomenlarged 10pt) randomized 4pt ;
   fill p withcolor lightgray ;
   draw p withcolor \MPvar{linecolor}
    withpen pencircle scaled \MPvar{linewidth};
   endfor;
\stopuseMPgraphic
```

Now, we want hook this frame to the definition of an axiom. We can use

```
\defineenumeration
   [axiom]
   [    text=Axiom,
      before={\startaxiomframe},
       after={\stopaxiomframe},
       title=yes,
     stopper=,
    location=top,
        list=all,
    listtext={Axiom }]
```

after which

```
\startaxiom {Playfair’s axiom}
   Exactly one line can be drawn through any point not on a given line
   parallel to the given line.
\stopaxiom
```

gives

Axiom 1 (Playfair’s axiom)

Exactly one line can be drawn through any point not on a given line parallel to the given line.

We can use all the fancy features of framed texts to get fancy backgrounds for enumerations.

Sharing styles

Suppose we spent a lot of effort in defining a fancy style for axioms. Now, suppose we want to define two new enumerations, postulate and proposition, with the same style. We want postulate and axioms to share numbering, but we want propositions to be numbered on their own. We could copy the entire style of axiom for defining postulates and propositions, but ConTEXt provides an easier method. We can define postulates as

```
\defineenumeration
  [postulate]
  [axiom]
  [    text=Postulate,
      list=all,
   listtext={Postulate }]
```

and then use `\startpostulate ...\stoppostulate` to get

Postulate 2 (Parallel Postulate)

If two lines are drawn which intersect a third in such a way that the sum of the inner angles on one side is less than two right angles, then the two lines inevitably must intersect each other on that side if extended far enough.

Notice that postulates have the same style as axioms, and they also share the numbering with axioms. Now, if we do not want to share the number, but still want to share the style we can say

```
\defineenumeration
  [proposition]
  [axiom]
  [    text=Proposition,
     number=proposition,
       list=all,
   listtext={Proposition }]
```

and then use `\startproposition ...\stopproposition` to get

Proposition 1

To construct an equilateral triangle on a given finite straight line.

Notice that the proposition has got a number of its own. We could have also used the `number` key to share the number with an already defined enumeration.

List of things

Often we want to display a list of theorems. For example, here is a list of all theorems used in this article.

Theorem 1	Pythagoras Theorem	27
Theorem 2		28
Corollary 3		29
Axiom 1	Playfair's axiom	30
Postulate 2	Parallel Postulate	31
Proposition 1		31

We get this by keying in

```
\placelist[enumeration:all][width=6em,criterium=all]
```

Observe that we had used `list=all` in all the enumerations we defined. We also set the `listtext` to appropriate strings. The list of theorems in stored in `enumeration:all` list, and can be recalled by `\placelist`. The `criterium=all` is needed so that we can place the theorems from all sections, not just from the current section.

Conclusion

Enumerations take care of most of the features needed for typesetting theorems. There are a few things that need improvement. It would be nice to have an automated placement of close symbol that works correctly with formulas and itemizations. The `ntheorem` package in LaTeX shows how it can be done using a two pass mechanism. I personally am not too happy with the way lists work; it would be nice if the list text was equal to text by default (rather than the name of the enumeration which starts with a lowercase letter) and list width was calculated automatically.

Aditya Mahajan
`adityam@umich.edu`

Exam Papers
Posing Questions To Students

Abstract
Exam is a module for consistent production and maintenance of student examinations. Provided for are various types of questions such as with long and small answers, yes/no questions and multiple choice.

Keywords
examination, multiple choice, ConTeXt

Introduction

The ConTEXt *hvdm-exm* module is a means for easy and consistent typesetting of exams and collections of exam questions. It especially facilitates questions with short answers and multiple choice. Many aspects of the typesetting are configurable.

This module is an adapted and upgraded version of my previous LaTEX package called the *exam* package. Its last version 3.30 of 1997/03/14 should still be available on CTAN (`macros/latex/contrib/exams`) but is no longer maintained by me. Changing to ConTEXt proved an efficient route to enhanced behaviour.

Exam structure

Exams can be typeset in two different formats. First of course in the format as presented to students undergoing the examination; this version can be typeset with nothing more than the bare examination questions. The other extreme is a collection of all questions with answers, annotations and points awarded. In between these is a lot of flexibility.

Customization may be done by setting various parameters, either specific to the macros in this module or through those inherent to the ConTEXt macros on top of which they are built. As a last resort one can redefine macros at will; there is a fair amount of modularisation to facilitate this.

A small builtin vocabulary of common language dependent terms is used. Defining them for another language allows one to typeset these terms automatically if that is ConTEXt's current active language. Looking at the code it will be apparent what is needed for additional languages.

All typesetting of exams, either for an examination or for the whole collection, has the following simple overall structure:

```
\startexam[options]
  \setexamdirectory[first series]
  \question[options]{file-1}
  \question[options]{file-2}
  ...
  \setexamdirectory[second series]
  \question[options]{file-3}
  \question[options][buffer]{\getbuffer}
  ...
\stopexam
```

Each question resides in its own file, but these may be put in different directories. With `\setexamdirectory` the reading directory is switched. One can intersperse the typesetting with all sort of text material. As an alternative to file reading, the questions can also come from ConTEXt buffers; this makes their inclusion in expository texts easy. The last question in the example above uses text defined between ConTEXt's simple `\startbuffer ... \stopbuffer` construct. If its second parameter is `buffer` the argument within the braces is interpreted as the callup of a buffer instead of a filename. The questions themselves are defined with:

```
\startquestion[options]
  ... text question followed by answer ...
\stopquestion
```

At the end of the exam a titlepage is made. Typesetting this after the questions makes for easy production of a summary with data such as the number of questions and the total number of points to be earned. For old hands: typesetting the titlepage after the contents was already seen in the early days of TEX processing, when program text was done with the *webmac* macro package.

Filenames may omit the `.tex` suffix, as is usual in the TEX world. Without an explicit `\setexamdirectory` the file is taken from the current directory.

More often then not — at least in my experience — one mistypes the name of some file in a long series of them. Instead of aborting the production run, or worse forgoing the mistake silently, there is a prominent message drawing attention to the mistake. Below is the warning that `missing-file.tex` wasn't located:

FILE missing-file UNKNOWN

Examples

Following are some examples of the capabilities of the module.

Short answer. First a simple question where a short answer has to be given. The definition is here:

```
\startquestion[%
  date=23-07-2006,
  score=2,
  subject={Short answer}]
What is the most famous question?
\shortanswer To be or not to be.\par
\stopquestion
```

The arguments of the `\startquestion` serve the following purpose. The `date` field is for the creation date of the question. The `score` is the maximum attainable number of points that can be earned by giving a correct answer (though this value may be overridden at production of the exam, if so required). The `subject` provides a short description for the question. The body of the question contains its text, followed by the predeterminded answer. Being a fairly short answer the `\shortanswer` form is chosen; note the `\par` that must finish it. If called with option `showanswer=no` (a natural choice for the production of a student copy) the typeset question looks like this:

Problem 1:	What is the most famous question?
Answer:	....

The answer has to be filled in on the dotted line. Setting the option to `showanswer=yes` in the option field of `\startexam` lets the dotted line disappear and the answer appears instead.

Problem 1:	What is the most famous question?
Answer:	To be or not to be.

There is an option (not demonstrated here) for showing the scorebox: in the margin a square is placed, containing the maximum score set for the question. The size of the square is easily changed by redefinition: `\def\scoreboxsize{dimension}`.

Below one can observe the extra information at the top of the question for the `series=yes` option.

—— *Short answer $***$ file = buffer : 23-07-2006* ——

Problem 1:	What is the most famous question?
Answer:	To be or not to be.

In the next examples the output is given twice, with respectively `showanswer=no` and `yes`. Furthermore the parameters date, score and subject of the `\startquestion` are omitted from the accompanying code as shown.

Standard answer. Opposed to the short answer stands the standard answer. The body of the answer is enclosed in the pair `\startanswer` ... `\stopanswer`. Enough vertical room must be reserved to allow the student to give a complete answer. That space is specified on the definition of the question in the first option argument; it can however be overriden when necessary. Omitting it takes the last default that has been set with option `answerspace`. In case answers are produced, the vertical space parameter is ignored and the answer will be typeset in its natural height.

```
\startquestion
\annotation Short question, long answer.\par
Elaborate on the question
\quotation{To be or not to be}.
\startanswer[2cm]
... sample answer text ...
\stopanswer
\annotation A second note.\par
\stopquestion
```

Problem 2:	Elaborate on the question “To be or not to be”.
Answer:	

In the next printout, containing the answer, the `notes` option has been set to show the `\annotation` macro. Annotions are useful as a form of comment, being neither part of the question nor of the answer. Annotations are collected and typeset together at the end of the question–answer body. If more than one annotation is present, they are automatically itemized (see also the note in example 6).

Problem 2: Elaborate on the question “To be or not to be”.
Answer: This is a famous quote from *Hamlet*, a play of the English writer William Shakespeare. One really has to suspect that nowadays not many students have seen even one of Shakespeare’s plays, let alone having read one. Most might not even know when or where William Shakespeare is supposed to have been born.

—— *Notes* ——
1. Short question, long answer.
2. A second note.

There is a third possibility in the way a long answer is treated; it is illustrated in the code below. The option parameter of `\startanswer` is given the value `force`. As a result the answer block is typeset, regardless whether answer showing is on or off. This is useful in those cases where a template must be filled in by the student. The `\ifanswers..\fi` macro allows one to show or hide the answer parts.

```
\startquestion
Finish the following ..
\startanswer[force]
\startitemize[2]
\item this .. \ifanswers Hamlet\fi
\item of .. \ifanswers William Shakespeare\fi
\item .. at \ifanswers Stratford-on-Avon\fi
\stopitemize
\stopanswer
\stopquestion
```

Problem 3: Finish the following three statements on the quotation “To be or not to be”.
Answer:
– this is a famous quote from
– of the English writer
– supposed to be born at

Problem 3: Finish the following three statements on the quotation “To be or not to be”.
Answer:
– this is a famous quote from Hamlet
– of the English writer William Shakespeare
– supposed to be born at Stratford-on-Avon

Alternating answer. For questions that can be answered simply by crossing out or underlining one of two alternatives (such as yes/no, right/wrong) there is the `\altanswer` construct. Its two arguments carry the alternatives. In answer mode the correct one is underlined; its position being the first one by default, to be changed with the option. The coding is:

```
\startquestion
Was something rotten in the state of Denmark?
\altanswer[left]{\Yes}{\No}
\stopquestion
```

Problem 4: Was something rotten in the state of Denmark?
Answer: yes *or* no

Problem 4: Was something rotten in the state of Denmark?
Answer: $\underline{\text{yes}}$ *or* no

Boxed answer. Sometimes an answer is just one thing to put inside a box. The `\answerbox` macro is made for this type of question. The length of the box can be specified in the optional argument; omitting it defaults to the natural width of the answer text. As with the long answer, here too one can change the default width examwide; the relevant option is `boxwidth`. The example:

```
\startquestion
Who posed that question?
\answerbox[3cm]{\quotation{Hamlet}}
\stopquestion
```

Problem 5: Who posed that question?
Answer: $\boxed{\phantom{\text{“Hamlet”}}}$

Problem 5: Who posed that question?
Answer: $\boxed{\text{“Hamlet”}}$

Parametrized question. When calling up a question it is possible to customize it by means of a parameter. To choose for example between several alternative formulations of essentially the same question. This helps to vary successive exams, while still keeping them similar. The parameter is given its default value on the `\startquestion` with the `parameter=value` option and can be redefined on the `\question` call. An explanation of parameter usage might come in handy and may find its place inside an annotation. In the example below `\question[parameter=2]` is used to select the second alternative whereas 1 is the default.

```
\startquestion[parameter=1]
Was something rotten in the state of
\ifx\parameter\parameterdefault
      Denmark? \altanswer[left]{\Yes}{\No}
\else Britain? \altanswer[right]{\Yes}{\No}
\fi
\startannotation ... \stopannotation
\stopquestion
```

Problem 6:	Was something rotten in the state of Britain?
Answer:	yes *or* <u>no</u>

—— *Notes* ——

The actual value 2 of `\parameter` is used; the default `\parameterdefault` has value 1.

Multiple choice question. Finally an example of a multiple choice question. In multiple choice there is no inviting “Answer” prompt, because the intention is obvious. When the necessary modules for random generation (*hvdm-rng*) and list sorting (*hvdm-lst*) are installed, one may automatically shuffle the items. The itemlist appearance is determined by the `setupmc` option; there one can give a `\setupitemize` to be executed just before the typesetting of the itemlist.

```
\startquestion
Which person is found in \quote{Hamlet}?
\startmultiplechoice
\wrong Rosalind\par
\wrong Desdemona\par
\right Ophelia\par
\wrong Juliet\par
\stopmultiplechoice
\stopquestion
```

Problem 7: Which person is found in ‘Hamlet’?

☐ Rosalind
☐ Desdemona
☐ Ophelia
☐ Juliet

Macros and their parameters

A summary of the most important macros and their arguments follows.

`\setupexams[..,..=..,..]`. Does the setup for the subsequent `\exam`’s. The options are summarized in the table. Because questions are put inside a `framedtext`, their appearance can be influenced with `\setupframedtexts[]`. Some `framedtext` parameters can be set from the option list of `\exam` and `\question`. These options are `rulethickness`, `offset`, `frameoffset`, `radius`, `corner`, `framecolor`, `background`, `backgroundoffset`, `backgroundscreen`, `backgroundcolor`, `backgroundcorner`, `backgroundradius`.

`\startexam[..,..=..,..]...\stopexam`. Specify an exam between this start and stop pair. The same options as with `\setupexams` apply here. Special values to mention are:

- ▷ `language` to choose the language in which to typeset the fixed terms;
- ▷ `marks` to choose the marks for multiple choice, customize these yourself by redefinition of `\rightmark`, `\wrongmark` and `\choosemark`;
- ▷ `numbering` switches numbering on/off;
- ▷ `prompt` set this option to `no` in order to suppress typesetting the “Answer” prompts for exams to be answered on separate answer sheets;
- ▷ `random` for random processing and checking with `\ifrandom..\else..\fi` when module *hvdm-rng* is installed, for randomization of multiple choice install *hvdm-lst* too;
- ▷ `separator` separation of questions with hairline, nothing or placement on separate page;
- ▷ `setupmc` setup for multiplechoice itemlist;
- ▷ `showanswer` for typesetting the answers;
- ▷ `showscore` for showing the score values;
- ▷ `standalone` typesets each question apart as is done in this text;
- ▷ `series` produces a catalogue of questions.

`\setexamdirectory[path]`. Set current directory to the (possibly empty) value of the argument.

`\setnumberstart[number]`. Sets the initial value for the numbering of questions. The numbering starts at 1 by default, but one can imagine numbering schemes where different topics each start at a round number. The option value `reset` restarts the numbering at the beginning.

`\question[..=..]{filename}`. Callup question from a file. The available options are those marked Q in the table. It is possible to include frame parameters in the option list for individual changes. The `number=#` option allows one to have questions individually numbered. Note however that the regular number is still advanced by 1.

`\question[..=..][buffer]{\getbuffer}`. Get the question from buffer instead of from file.

after	QE	*command*	execute after question
answerspace	DQE	*dimension* force	set default answer space
before	QE	*command*	execute before question
boxwidth	QE	*dimension*	set default width answerbox
date	DE	*text*	examination (E) or creation (D) date
frame	QE	on off	put frame around question
hang	QE	*number*	hang parameter on question and answer
introstyle	QE	*stylecommand* normal	fontstyle of question, answer intro
language	E	*2-letter* currentlanguage	languagecode for fixed vocabulary
marks	E	square circle squarev circlev	multiplechoice marks
notes	QE	yes no	place annotations
numbering	E	yes no	show question numbering
parameter	QD	*any*	actual (Q) or default (D) parameter value
prompt	QE	yes no	place answer part or always suppress
random	QE	yes no	random processing turned on or off
score	QD	*number*	the question score
separator	E	yes no page	separator or pagebreak after question
series	E	yes no	typeset catalogue of questions
setupmc	QE	`\setupitemize`	format multiplechoice item list
showanswer	E	yes no	show answers
showscore	E	yes no	show score and possibly subscores
standalone	E	yes no	typeset isolated question
style	QE	*stylecommand* normal	fontstyle of question body
stopper	QE	*char*	placed behind question and answer
subject	E	*text*	field of the exam
width	QE	fit broad *dimension*	width of question

Setups effective on question definition (D), question (Q) exam (E)

`\startquestion[..=..]...\stopquestion`. Put the definition of a question fully inside this macro pair. The most obvious pattern is to start with the text of the question and let this follow by the answer. Typesetting begins with the "Question" header and an optional sequence number, any `answer` macro following then triggers the "Answer" header. The options applicable are summarized below.

date	*text*	empty	creation date
subject	*text*	empty	descriptive text
score	*number*	0	points allotted
parameter	*any*	empty	parameter default

When a question has more than one part, it is handy to allot a fixed number of points to each subquestion. The macro `\subscore[#]` places them in the margin when `showanswer=yes`. It helps maintaining consistency by reporting an error if the sum of the subscores does not equal the score given to the question as a whole. The `number` option makes it possible to number each question individually or reset the numbering from a given question on.

`\shortanswer...\par`. Answer text of one paragraph to be placed on a dotted line. The closing `\par` that may not be omitted.

`\startanswer[length]...\stopanswer`. Answer text is put inside this pair. The optional `length` parameter specifies the vertical amount of space to reserve for the answer. If it is not given the default value is used. Set the length parameter to `force` in order to force typesetting of the answer block in its natural height unless suppressed completely with the `prompt=no` setting. The value given here always prevails, because it is clearly something the designer of the question had in mind. It is therefore advised to use this parameter sparingly and preferably tune these lengths on `\startexam` (global) and/or on the `\question` (individual).

`\answerbox[width]{...}`. Answer to be placed inside a box of given width. If width is not given the default value is used. The default width can be specified by the `boxwidth` parameter.

`\altanswer[left,right]{1st}{2nd}`. Two alternating answers. Defaults to the first alternative as the correct one, any option value other than `left` or empty switches this to the second argument. Redefine macro

\altanswerseparator for the separator between the two possibilities.

\startmultiplechoice...\stopmultiplechoice \right...\par, \wrong...\par. A multiple choice item list should be put inside the start/stop pair. Place the correct item after \right and the wrong items after \wrong, closing them with \par; if you like a synonym like \ok better, just define \ok as \right. The symbols marking the items can be chosen with the marks option on \startexam.

\annotation ... \par \startannotation...\stopannotation. If put inside the question, the annotations will be added at the end of it according to the value of the notes option. For example useful to explain \parameter usage. The first format limits the contents to one paragraph; do not forget the closing \par then.

\examtitlepage. Produces a standard titlepage. Redefine this macro at will. Macros \thetitle, \thedate, \thetotalquestions, \thetotalscore are available for use; the latter two are filled in by \stopexam.

\endefines. English language vocabulary. Others may be added by defining a macro of the form \xxdefines similar to \endefines. Showing the vocabulary for a language can be done with \meaning\xxdefines.

Hans van der Meer
hansm@science.uva.nl

Revision control for TEX documents
An overview

Abstract
Revision control is the management of multiple versions of the same unit of information. Originating in formalized processes in engineering, it was first automated for managing source code for computer software. Since TEX documents are like source code, they lend themselves well to being managed by a revision control system.
Systems like `RCS` and `git` are very suitable for single writers working on their own projects.
More elaborate systems like `CVS` and `subversion` are more suited for groups cooperating on projects. It takes more effort to master them.
For most single users, `git` is the best alternative for multi-file projects, followed by `RCS` for working on single TEX files.

Keywords
revision control, RCS, CVS, subversion, git

Introduction

What is revision control

Revision control is keeping a history of the development of a unit of information.

The first revision control systems were procedures used by draftsmen to differentiate between several versions of drawings or blueprints. Engineering drawings are customarily equipped with a table in the lower right corner, stating the date of a revision, a revision number and a description of the changes since the previous version. While these procedures are helpful, one also had to make a copy of the master drawing before it was changed to keep a full history of the evolution of the drawing.

With the rise of software engineering, similar procedures for tracking the development of computer source code were automated. Programs were written to facilitate storing and retrieving different versions of (plain text) source code files. Changes are usually saved as differences (or diffs) with respect to the previous version, to conserve space.

Since TEX files are plain text files, they lend themselves very well to being managed by revision control systems.

Newer and more sophisticated revision control systems can also handle binary files, like PDF and JPG files efficiently, making them suitable for complex TEX projects.

Modern revision control systems enable several people to work together on a project. This article however will focus on a single person using revision control, since that is envisioned as the most common scenario.

This article focuses on software that is freely available. There are several proprietary systems available but they are usually quite expensive and therefore out of reach of most single users. And as the freely available systems mentioned here are used to manage most of the largest open-source software projects in existence, they are good enough.

Why use revision control

Revision control is primarily useful for maintaining documents that are long-lived and frequently edited. By using a revision control system one can:

- Undo edits, especially deletions.
- Track the history of a document; what has changed and why was it changed. And for documents edited by more than one person; who made the change.
- Work on a single project from multiple machines or with more persons.
- Merge different versions of a document.

Most editors offer an undo option. But this is usually limited to the edits done in the current session. With a revision control system one can easily restore a document or project to an earlier state.

And while it is possible to record the history of a document e.g. in a comment at the start of a file, this requires a lot of discipline and is easily forgotten. Besides, sometimes the record is just not enough; you may want to retrieve an earlier version of a document or project.

A simple system

Probably the most simple system of revision control is regularly copying a file you are working on under a different name, e.g. with the date in the filename. So if you are working on a file called `foo.tex` you could store a copy of it at the end of the day as `foo-YYYYMMDD.tex`. The date is embedded in the filename to make it unambiguous and unique. If your project has lots of files, you can wrap them up in a similarly named zip-file or tarball.

This system has the advantage of being simple, since it does not rely on auxiliary programs. As with any other revision control system, it does depend on operator discipline. It is usable on all operating systems.

However, for large projects this system can use an awful lot of storage. One of the author's projects is around 24 MiB[1] in size. Copying that every day would waste a lot of disk space. And it does not facilitate several persons working on a single document or project. Nor does it make it easy to see what has changed between different versions.

RCS

This system was developed in the 1980s by Walter Tichy at Purdue University, as a free and better replacement for the proprietary SCCS.[2] This set of programs deals with single text files. It does not handle binary files well, and it has no concept of a project. It is therefore primarily suited for projects with a limited number of text files. RCS originated as a command-line set of programs for UNIX. It has been ported to almost every UNIX-like system available. A Windows version is also available.[3]

In the following examples the command-line versions of the RCS tools will be used. These examples are not meant to be a replacement for the manual pages that come with the RCS suite, or for the plethora of HOWTO documents available on the Internet.[4,5] A quick search with your favorite search engine for "RCS HOWTO" will turn up a lot of links.

These examples are meant to give you a taste of how a revision control system works. Other systems have different but functionally similar commands.

After creating a new file, it has to be put under control of RCS. This is done with the `ci` command:

```
$ ci foo.tex
RCS/foo.tex,v  <--  foo.tex
enter description, terminated with single '.'
NOTE: This is NOT the log message!
>> Test file.
>> .
initial revision: 1.1
done
```

RCS now asks you to give a description of the file. After giving that, the file is checked in.

A file called `foo.tex,v` has now been created that contains the revision history of the file `foo.tex`. If a subdirectory named `RCS` exists, the revision history file will be put there. Otherwise it will be put in the same directory as the original file. The initial check-in is given the revision number 1.1.

To edit the file, it has to checked out and locked with the `co` command:

```
$ co -l foo.tex
RCS/foo.tex,v  -->  foo.tex
revision 1.1 (locked)
done
```

RCS uses locking to make sure that only one person is editing a file at a given time. This can prove awkward in an environment where multiple persons are working on a project. You cannot check in your changes while someone else has locked the file. And you might undo his changes by checking in yours! Other systems handle this better.

Having checked out the file, you can now use your favorite editor to edit it. Some editors like `emacs` and `vim` are aware of RCS files and have special menus or commands that enable you to check files in and out from the editor.

You can use the `rcsdiff` command to check what has changed in the working file with respect to the last checked in version:

```
$ rcsdiff -u foo.tex
============================================
RCS file: RCS/foo.tex,v
retrieving revision 1.1
diff -u -r1.1 foo.tex
--- foo.tex 2007/07/24 16:00:50 1.1
+++ foo.tex 2007/07/24 18:15:08
@@ -1,3 +1,7 @@
-bla
+
+foo
+
+bar
+
 \bye
```

The `-u` option of the `rcsdiff` command selects the so-called "unified diff" format. In the author's opinion this is the easiest to read, because it provides some context to the changes. As you can see, RCS uses lines as the smallest unit. So if one letter changes in a line, the whole line is seen as changed. Lines that are removed are preceded with a "-" in unified diff format, while added lines are preceded with a "+". The line-based difference mechanism used by RCS is what makes it difficult to use with binary files, since non-text files usually do not have meaningful regular line breaks.

If you are finished editing for the day, or if you feel that you have reached a stage in your document that you want to save, you check in the changes:

```
$ ci -u foo.tex
RCS/foo.tex,v  <--  foo.tex
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.'
>> This is the log message.
>> .
done
```

Using the `-u` option of the `ci` command enables you to go on editing directly. Without it you would have to check out and lock the file again.

CVS

CVS[6] was developed from RCS by Dick Grune in the mid-1980s to handle projects consisting of multiple files in directory trees instead of single files.[7]

Unlike RCS, it can use a client-server architecture and it separates the place where the revision history is stored (called the repository) from the working directory. The server (and the repository) do not have to reside on the same machine as the client (and working directory), making collaboration over a network easier.

The downside is that it is more complicated to work with than RCS. CVS cannot attach a revision to moving or renaming a file or directory. CVS uses a centralized server if multiple persons are working together on a project. For most single-user TEX projects, this will be an overweight solution. The added features of CVS come at the price of added complexity. And since it is based on RCS, it still doesn't handle binary files well.

Many open source software projects use CVS to maintain their code.

Subversion

Advertised as "CVS done right", subversion[8] aims to be a compelling replacement for CVS. This system solves a lot of the problems of CVS; it handles binary files and it can handle file and directory renames. But like CVS it is probably overkill for a small project. And it shares with CVS a steep learning curve, and the need for separate repositories.

Git

Git[9] was developed when the developers of the Linux kernel lost access to the proprietary BitKeeper system[10]. Development was started in April 2005 with version 1.0 being released in December 2005. The Linux kernel development has been managed with git since June 2005.

Unlike RCS, git tracks a directory tree of files. It also handles binary files. But like RCS, a git-managed directory is completely self-contained. It does not require external repositories. It has few dependencies, stores its data in compressed form and is quite fast. For instance, the complete revision history of the 24 MiB project mentioned before which spans two years and 99 revisions is only 12 MiB. It is also a distributed system. Every directory (repository) is self-sufficient, but synchronizing between them is easy. So it is both easy to work with for a stand-alone user, and for a group of people working on the same project.

Because of its lightweight nature, directory tracking and handling of binaries, the author now prefers it over using RCS.

An example of working with git follows. In this article it is impossible to showcase the complete functionality of git, since there are more than a hundred and forty git commands. Manual pages for all the git commands, as well as several tutorials and HOWTOs are included in the git distribution[6].

To create a git repository, change to the directory whose contents you want to monitor, and type:

```
$ git-init-db
Initialized empty Git repository in .git/
```

This creates a `.git` subdirectory that git uses to store all data.

The `git status` command shows the state of the repository.

```
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include
in what will be committed)
#
# Makefile
# rc.tex
nothing added to commit but untracked
files present (use "git add" to track)
```

Now one has to point out which files you want to monitor:

```
$ git add Makefile rc.tex
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
# new file: Makefile
# new file: rc.tex
#
```

To save this initial state of the files, commit them:

```
git commit -a \
-m "Initial check-in of Makefile and rc.tex."
Created initial commit 9d3de62:
Initial check-in of Makefile and rc.tex.
 2 files changed, 357 insertions, 0 deletions
 create mode 100644 Makefile
 create mode 100644 rc.tex
```

The `-a` flag indicates that all changes should be committed, while the `-m` flag is followed by the commit message.

With the `git log` command one can list the commits:

```
$ git log
commit 9d3de6265ad4fc0df2ca108b7918f6283dc18d59
Author: Roland Smith <rsmith@slackbox.xs4all.nl>
Date:   Thu Jul 26 19:03:57 2007 +0200

    Initial check-in of Makefile and rc.tex.
```

Every commit is identified by a SHA1 hash of its contents. This is also used to check against data corruption. Next is the name and e-mail address of the person who checked in this commit. The date of the commit and the commit message are also listed.

If one compiles the TEX file, some extra files are created that git doesn't know about:

```
$ git status
# On branch master
# Changed but not updated:
#
# modified:   rc.tex
#
# Untracked files:
#
# rc.aux
# rc.log
# rc.pdf
no changes added to commit
```

Normally, we would like git to ignore those files. So we add them to the text file `.git/info/exclude`, and git will ignore them from now on.

The command `git diff` is used here to show the changes in the monitored files since the last commit. A small piece of the diff is shown below. It uses the so-called unified diff output style.

```
$ git diff
diff --git a/rc.tex b/rc.tex
index e3eaefe..3aac05d 100644
--- a/rc.tex
+++ b/rc.tex
@@ -1,5 +1,5 @@
 % -*- latex -*-
-% Time-stamp: <2007-07-26 19:01:53 rsmith>
+% Time-stamp: <2007-07-26 19:20:47 rsmith>
 % Copyright © 2007 R.F. Smith <rsmith@xs4all.nl>
 %
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
@@ -292,10 +292,11 @@ to work with for a stand-al
 the same project.

 Because of its lightweight nature, directory
 tracking and handling of binaries, the
-author now prefers it over using RCS.
+author now prefers it over using RCS. An
+example of working with git follows.
...
```

By supplying `git diff` with one or two commit-id's, you can also show the differences between the named commit and the working files, or between two named commits.

Conclusion

For documents that consist of a single or a small number of TEX files and other text files, RCS is probably the best solution because of its simplicity.

For larger projects, git is a better option. Especially if they contain binary files in e.g. PDF or JPG format that need to be controlled. Or if you want to collaborate with others on a project.

Systems like CVS or subversion will probably appeal to larger organizations that require centralized development.

Notes

1. http://en.wikipedia.org/wiki/MiB
2. http://en.wikipedia.org/wiki/Revision_Control_System
3. http://www.codeproject.com/tools/cs-rcs.asp
4. http://www.madboa.com/geek/rcs/
5. http://www.athabascau.ca/html/depts/compserv/webunit/HOWTO/rcs.htm
6. http://www.nongnu.org/cvs/
7. http://en.wikipedia.org/wiki/Concurrent_Versions_System
8. http://subversion.tigris.org/
9. http://git.or.cz/
10. http://en.wikipedia.org/wiki/Git_(software)

Roland Smith
rsmith@xs4all.nl

The luafication of TEX and ConTEXt

Introduction

Here I will present the current stage of LuaTEX around beta stage 2, and discuss the impact so far on ConTEXt MkIV that we use as our testbed. I'm writing this at the end of February 2008 as part of the series of regular updates on LuaTEX. As such, this report is part of our more or less standard test document (`mk.tex`). More technical details can be found in the reference manual that comes with LuaTEX. More information on MkIV is available in the ConTEXt mailing lists, Wiki, and `mk.pdf`.

For those who never heard of LuaTEX: this is a new variant of TEX where several long pending wishes are fulfilled:

- combine the best of all TEX engines
- add scripting capabilities
- open up the internals to the scripting engine
- enhance font support to OpenType
- move on to Unicode
- integrate MetaPost

There are a few more wishes, like converting the code base to c but these are long term goals.

The project started a few years ago and is conducted by Taco Hoekwater (Pascal and c coding, code base management, reference manual), Hartmut Henkel (pdf backend, experimental features) and Hans Hagen (general overview, Lua and TEX coding, website). The code development got a boost by a grant of the Oriental TEX project (project lead: Idris Samawi Hamid) and funding via the tug. The related MPlib project by the same team is also sponsored by several user groups. The very much needed OpenType fonts are also a user group funded effort: the Latin Modern and TEX Gyre projects (project leads: Jerzy Ludwichowski, Volker RW Schaa and Hans Hagen), with development (the real work) by: Bogusław Jackowski and Janusz Nowacki.

One of our leading principles is that we focus on opening up. This means that we don't implement solutions (which also saves us many unpleasant and everlasting discussions). Implementing solutions is up to the user, or more precisely: the macro package writer, and since there are many solutions possible, each can do it his or her way. In that sense we follow the footsteps of Don Knuth: we make an extensible tool, you are free to like it or not, you can take it and extend it where needed, and there is no need to bother us (unless of course you find bugs or weird side effects). So far this has worked out quite well and we're confident that we can keep our schedule.

We do our tests of a variant of ConTEXt tagged MkIV, especially meant for LuaTEX, but LuaTEX itself is in no way limited to or tuned for ConTEXt. Large chunks of the code written for MkIV are rather generic and may eventually be packaged as a base system (especially font handling) so that one can use LuaTEX in rather plain mode. To a large extent MkIV will be functionally compatible with MkII, the version meant for traditional TEX, although it knows how to profit from XƎTEX. Of course the expectation is that certain things can be done better in MkIV than in MkII.

Status

By the end of 2007 the second major beta release of LuaTEX was published. In the first quarter of 2008 Taco would concentrate on MPlib, Hartmut would come up with the first version of the image library while I could continue working on MkIV and start using LuaTEX in real projects. Of course there is some risk involved in that, but since we have a rather close loop for critical bug fixes, and because I know how to avoid some dark corners, the risk was worth taking.

What did we accomplish so far? I can best describe this in relation to how ConTEXt MkIV evolved and will evolve. Before we do this, it makes sense to spend some words on why we started working on MkIV in the first place.

When the LuaTEX project started, ConTEXt was about 10 years in the field. I can safely say that we were still surprised by the fact that what at first sight seems unsolvable in TEX somehow could always be dealt with. However, some of the solutions were rather tricky. The code evolved towards a more or less stable state, but sometimes depended on controlled processing. Take for instance backgrounds that can span pages and columns, can be nested and can have arbitrary shapes. This feature has been present in ConTEXt for quite a while, but it involves an interplay between TEX and MetaPost. It depends on information collected in a previous run as well as processing of graphics.

This means that by now ConTEXt is not just a bunch of TEX macros, but also closely related to MetaPost. It also means that processing itself is by now rather controlled by a wrapper, in the case of MkII called TEXexec. It may sound complicated, but the fact that we have implemented workflows that run unattended for many years and involve pretty complex layouts and graphic manipulations demonstrates that in practice it's not as bad as it may sound.

With the arrival of LuaTEX we not only have a rigourously updated TEX engine, but also get MetaPost integrated. Even better, the scripting language Lua is not only used for opening up TEX, but is also used for all kind of management tasks. As a result, the development of MkIV not only concerns rewriting whole chunks of ConTEXt, but also results in a set of new utilities and a rewrite of existing ones. Since dealing with MkIV will demand some changes in the way users deal with ConTEXt I will discuss some of them first. It also demonstrates that LuaTEX is more than just TEX.

Utilities

There are two main scripts: luatools and mtxrun. The first one started as a replacement for kpsewhich but evolved into a base tool for generating (tds) file databases and generating formats. In MkIV we replace the regular file searching, and therefore we use a different database model. That's the easy part. More tricky is that we need to bootstrap MkIV into this alternative mode and when doing so we don't want to use the `kpse` library because that would trigger loading of its databases. To discuss the gory details here might cause users to refrain from using LuaTEX so we stick to a general description.

- When generating a format, we also generate a bootstrap Lua file. This file is compiled to bytecode and is put alongside the format file. The libraries of this bootstrap file are also embedded in the format.
- When we process a document, we instruct LuaTEX to load this bootstrap file before loading the format. After the format is loaded, we re-initialize the embedded libraries. This is needed because at that point more information may be available than at loading time. For instance, some functionality is available only after the format is loaded and LuaTEX enters the TEX state.
- File databases, formats, bootstrap files, and runtime-generated cached data is kept in a tds tree specific cache directory. For instance, OpenType font tables are stored on disk so that next time loading them is faster.

Starting LuaTEX and MkIV is done by luatools. This tool is generic enough to handle other formats as well, like mptopdf or Plain. When you run this script without argument, you will see:

```
version 1.1.1 - 2006+ - PRAGMA ADE / CONTEXT

--generate        generate file database
--variables       show configuration variables
--expansions      show expanded variables
--configurations  show configuration order
--expand-braces   expand complex variable
--expand-path     expand variable (resolve
                                      paths)
--expand-var      expand variable (resolve
                                 references)
--show-path       show path expansion of ...
--var-value       report value of variable
--find-file       report file location
--find-path       report path of file
--make or --ini   make luatex format
--run or --fmt=   run luatex format
--luafile=str     lua inifile (default is
                             <progname>.lua)
--lualibs=list    libraries to assemble
                                  (optional)
--compile         assemble and compile lua
                                     inifile
--verbose         give a bit more info
--minimize        optimize lists for format
--all             show all found files
--sort            sort cached data
--engine=str      target engine
--progname=str    format or backend
--pattern=str     filter variables
--lsr             use lsr and cnf directly
```

For the Lua based file searching, luatools can be seen as a replacement for mktexlsr and kpsewhich and as such it also recognizes some of the kpsewhich flags. The script is self contained in the sense that all needed libraries are embedded. As a result no library paths need to be set and packaged. Of course the script has to be run using LuaTEX itself. The following commands generate the file databases, generate a ConTEXt MkIV format, and process a file:

```
luatools --generate
luatools --make --compile cont-en
luatools --fmt=cont-en somefile.tex
```

There is no need to install Lua in order to run this script. This is because LuaTEX can act as such with the advantage that the built-in libraries are available

```
version 1.0.2 - 2007+ - PRAGMA ADE / CONTEXT

--script              run an mtx script
--execute             run a script or program
--resolve             resolve prefixed arguments
--ctxlua              run internally (using preloaded libs)
--locate              locate given filename

--autotree            use texmf tree cf.\ environment settings
--tree=pathtotree     use given texmf tree (def: 'setuptex.tmf')
--environment=name    use given (tmf) environment file
--path=runpath        go to given path before execution
--ifchanged=filename  only execute when given file has changed
--iftouched=old,new   only execute when given file has changed

--make                create stubs for (context related) scripts
--remove              remove stubs (context related) scripts
--stubpath=binpath    paths where stubs wil be written
--windows             create windows (mswin) stubs
--unix                create unix (linux) stubs

--verbose             give a bit more info
--engine=str          target engine
--progname=str        format or backend

--edit                launch editor with found file
--launch (--all)      launch files (assume os support)

--intern              run script using built-in libraries
```

Figure 1. mtxrun help information

too, for instance the Lua file system `lfs`, the zip file manager `zip`, the Unicode libary `unicode`, `md5`, and of course some of our own.

luatex	a Lua–enhanced TEX engine
texlua	a Lua engine enhanced with some libraries
texluac	a Lua bytecode compiler enhanced with some libraries

In principle `luatex` can perform all tasks but because we need to be downward compatible with respect to the command line and because we want Lua compatible variants, you can copy or symlink the two extra variants to the main binary.

The second script, mtxrun, can be seen as a replacement for the Ruby script texmfstart, a utility whose main task is to launch scripts (or documents or whatever) in a tds tree. The mtxrun script makes it possible to get away from installing Ruby and as a result a regular TEX installation can be made independent of scripting tools.

The help information is shown in figure 1. It gives an impression of what the script does: running other scripts, either within a certain tds tree or not, and either conditionally or not. Users of ConTEXt will probably recognize most of the flags. As with texmfstart, arguments with prefixes like `file:` will be resolved before being passed to the child process.

The first option, `--script` is the most important one and is used like:

```
mtxrun --script fonts --reload
mtxrun --script fonts --pattern=lm
```

In MkIV you can access fonts by filename or by font name, and because we provide several names per font you can use this command to see what is possible. Patterns can be Lua expressions, as demonstrated in figure 2.

```
mtxrun --script font  --list --pattern=lmtype.*regular

lmtypewriter10-capsregular   LMTypewriter10-CapsRegular   lmtypewriter10-capsregular.otf
lmtypewriter10-regular       LMTypewriter10-Regular       lmtypewriter10-regular.otf
lmtypewriter12-regular       LMTypewriter12-Regular       lmtypewriter12-regular.otf
lmtypewriter8-regular        LMTypewriter8-Regular        lmtypewriter8-regular.otf
lmtypewriter9-regular        LMTypewriter9-Regular        lmtypewriter9-regular.otf
lmtypewritervarwd10-regular  LMTypewriterVarWd10-Regular  lmtypewritervarwd10-regular.otf
```

Figure 2. Example of a `mtxrun --script font` run.

A simple

```
mtxrun --script fonts
```

gives:

```
version 1.0.2 - 2007+ - PRAGMA ADE / CONTEXT
                                   | font tools

--reload       generate new font database
--list         list installed fonts
--save         save open type font in raw table

--pattern=str  filter files
--all          provide alternatives
```

In MkIV font names can be prefixed by `file:` or `name:` and when they are resolved, several attempts are made, for instance non-characters are ignored. The `--all` flag shows more variants.

Another example is:

```
mtxrun --script context --ctx=somesetup
                                  somefile.tex
```

Again, users of TEXexec may recognize part of this and indeed this is its replacement. Instead of TEXexec we use a script named `mtx-context.lua`. Currently we have the following scripts and more will follow:

The `babel` script is made in cooperation with Thomas Schmitz and can be used to convert babelized Greek files into proper utf. More of such conversions may follow. With `cache` you can inspect the content of the MkIV cache and do some cleanup. The `chars` script is used to construct some tables that we need in the process of development. As its name says, `check` is a script that does some checks, and in particular it tries to figure out if TEX files are correct. The already mentioned `context` script is the MkIV replacement of TEXexec, and takes care of multiple runs, preloading project specific files, etc. The `convert` script will replace the Ruby script `pstopdf`.

A rather important script is the already mentioned `fonts`. Use this one for generating font name databases (which then permits a more liberal access to fonts) or identifying installed fonts. The `unzip` script indeed unzips archives. The `update` script is still somewhat experimental and is one of the building blocks of the ConTEXt minimal installer system by Mojca Miklavec and Arthur Reutenauer. This update script synchronizes a local tree with a repository and keeps an installation as small as possible, which for instance means: no OpenType fonts for pdfTEX, and no redundant Type1 fonts for LuaTEX and XƎTEX.

The (for the moment) last two scripts are `watch` and `web`. We use them in (either automated or not) remote publishing workflows. They evolved out of the eXaMpLe framework which is currently being reimplemented in Lua.

As you can see, the LuaTEX project and its ConTEXt companion MkIV project not only deal with TEX itself but also facilitates managing the workflows. And the next list is just a start.

context	controls processing of files by MkIV
babel	conversion tools for LaTEX files
cache	utilities for managing the cache
chars	utilities used for MkIV development
check	TEX syntax checker
convert	helper for some basic graphic conversion
fonts	utilities for managing font databases
update	tool for installing minimal ConTEXt trees
watch	hot folder processing tool
web	utilities related to automate workflows

There will be more scripts. These scripts are normally rather small because they hook into mtxrun which provides the libraries. Of course existing tools remain part of the toolkit. Take for instance ctxtools, a Ruby script that converts font encoded pattern files to generic utf encoded files.

Those who have followed the development of ConTEXt will notice that we moved from utilities written in Modula to tools written in Perl. These were later replaced by Ruby scripts and eventually most of them will be rewritten in Lua.

Macros

I will not repeat what is said already in the MkIV related documents, but stick to a summary of what the impact on ConTEXt is and will be. From this you can deduce what the possible influence on other macro packages can be.

Opening up TEX started with rewriting all io related activities. Because we wanted to be able to read from zip files, the web and more, we moved away from the traditional kpse based file handling. Instead MkIV uses an extensible variant written in Lua. Because we need to be downward compatible, the code is somewhat messy, but it does the job, and pretty quickly and efficiently too. Some alternative input media are implemented and many more can be added. In the beginning I permitted several ways to specify a resource but recently a more restrictive url syntax was imposed. Of course the file locating mechanisms provide the same control as provided by the file readers in MkII.

An example of reading from a zip file is:

```
\input zip:///archive.zip?name=blabla.tex
\input zip:///archive.zip?name=/path/blabla.tex
```

In addition one can register files, like:

```
\usezipfile[archive.zip]
\usezipfile[tex.zip][texmf-local]
\usezipfile[tex.zip?tree=texmf-local]
```

The last two variants register a zip file in the tds structure where more specific lookup rules apply. The files in a registered file are known to the file searching mechanism so one can give specifications like the following:

```
\input */blabla.tex
\input */somepath/blabla.tex
```

In a similar fashion one can use the `http`, `ftp` and other protocols. For this we use independent fetchers that cache data in the MkIV cache. Of course, in more structured projects, one will seldom use the `\input` command but use a project structure instead.

Handling of files rather quickly reached a stable state, and we seldom need to visit the code for fixes. Already after a few years of developing the first code for LuaTEX we reached a state of 'Hm, when did I write this?'. When we have reached a stable state I foresee that much of the older code will need a cleanup.

Related to reading files is the sometimes messy area of input regimes (file encoding) and font encoding, which itself relates to dealing with languages. Since LuaTEX is utf-8 based, we need to deal with file encoding issues in the frontend, and this is what Lua based file handling does. In practice users of LuaTEX will swiftly switch to utf anyway but we provide regime control for historic reasons. This time the recoding tables are Lua based and as a result MkIV has no regime files. In a similar fashion font encoding is gone: there is still some old code that deals with default fallback characters, but most of the files are gone. The same will be true for math encoding. All information is now stored in a character table which is the central point in many subsystems now.

It is interesting to notice that until now users have never asked for support with regards to input encoding. We can safely assume that they just switched to utf and recoded older documents. It is good to know that LuaTEX is mostly pdfTEX but also incorporates some features of Omega. The main reason for this is that the Oriental TEX project needed bidirectional typesetting and there was a preference for this implementation over the one provided by ε-TEX. As a side effect input translation is also present, but since no one seems to use it, that may as well go away. In MkIV we refrain from input processing as much as possible and focus on processing the node lists. That way there is no interference between user data, macro expansion and whatever may lead to the final data that ends up in the to-be-typeset stream. As said, users seem to be happy to use utf as input, and so there is hardly any need for manipulations.

Related to processing input is verbatim: a feature that is always somewhat complicated by the fact that one wants to typeset a manual about TEX in TEX and therefore needs flexible escapes from illustrative as well as real TEX code. In MkIV verbatim as well as all buffering of data is dealt with in Lua. It took a while to figure out how LuaTEX should deal with the concept of a line ending, but we got there. Right from the start we made sure that LuaTEX could deal with collections of catcode settings (those magic states that characters can have). This means that one has complete control at both the TEX and Lua end over the way characters are dealt with.

In MkIV we also have some pretty printing features, but many languages are still missing. Cleaning up the premature verbatim code and extending pretty printing is on the agenda for the end of 2008.

Languages also are handled differently. A major change is that pattern files are no longer preloaded but read in at runtime. There is still some relation between fonts and languages, no longer in the encoding but in dealing with OpenType features. Later we will do a more drastic overhaul (with multiple name schemes and such). There are a few experimental features, like spell checking.

Because we have been using utf encoded hyphenation patterns for quite some time now, and because ConTEXt ships with its own files, this transition probably went unnoticed, apart maybe from a faster format generation and less startup time.

Most of these features started out as an experiment and provided a convenient way to test the LuaTEX extensions. In MkIV we go quite far in replacing TEX code by Lua, and how far one goes is a matter of taste and ambition. An example of a recent replacement is graphic inclusion. This is one of the oldest mechanisms in ConTEXt and it has been extended many times, for instance by plugins that deal with figure databases (selective filtering from pdf files made for this purpose), efficient runtime conversion, color conversion, downsampling and product dependent alternatives.

One can question if a properly working mechanism should be replaced. Not only is there hardly any speed to gain (after all, not that many graphics are included in documents), a Lua–TEX mix may even look more complex. However, when an opened-up TEX keeps evolving at the current pace, this last argument becomes invalid because we can no longer give that TEXie code to Lua. Also, because most of the graphic inclusion code deals with locating files and figuring out the best quality variant, we can benefit much from Lua: file handling is more robust, the code looks cleaner, complex searches are faster, and eventually we can provide way more clever lookup schemes. So, after all, switching to Lua here makes sense. A nice side effect is that some of the mentioned plugins now take a few lines of extra code instead of many lines of TEX. At the time of writing this, the beta version of MkIV has Lua based graphic inclusion.

A disputable area for Luafication is multipass data. Most of that has already been moved to Lua files instead of TEX files, and the rest will follow: only tables of contents still use a TEX auxiliary file. Because at some point we will reimplement the whole section numbering and cross referencing, we postponed that till later. The move is disputable because in the end, most data ends up in TEX again, which involves some conversion. However, in Lua we can store and manipulate information much more easily and so we decided to follow that route. As a start, index information is now kept in Lua tables, sorted on demand, depending on language needs and such. Positional information used to take up much hash space which could deplete the memory pool, but now we can have millions of tracking points at hardly any cost.

Because it is a quite independent task, we could rewrite the MetaPost conversion code in Lua quite early in the development. We got smaller and cleaner code, more flexibility, and also gained some speed. The code involved in this may change as soon as we start experimenting with MPlib. Our expectations are high because in a bit more modern designs a graphic engine cannot be missed. For instance, in educational material, backgrounds and special shapes are all over the place, and we're talking about many MetaPost runs then. We expect to bring down the processing time of such documents considerably, if only because the MetaPost runtime will be close to zero (as experiments have shown us).

While writing the code involved in the MetaPost conversion a new feature showed up in Lua: `lpeg`, a parsing library. From that moment on `lpeg` was being used all over the place, most noticeably in the code that deals with processing xml. Right from the start I had the feeling that Lua could provide a more convenient way to deal with this input format. Some experiments with rewriting the MkII mechanisms did not show the expected speedup and were abandoned quickly.

Challenged by `lpeg` I then wrote a parser and started playing with a mixture of a tree based and stream approach to xml (MkII is mostly stream based). Not only is loading xml code extremely fast (we used 40 megaByte files for testing), dealing with the tree is also convenient. The additional MkIV methods are currently being tested in real projects and so far they result in an acceptable and pleasant mix of TEX and xml. For instance, we can now selectively process parts of the tree using path expressions, hook in code, manipulate data, etc.

The biggest impact of LuaTEX on the ConTEXt code base is not the previously mentioned mechanisms but one not yet mentioned: fonts. Contrary to XƎTEX, which uses third party libraries, LuaTEX does not implement dealing with font specific issues at all. It can load several font formats and accepts font data in a well-defined table format. It only processes character nodes into glyph nodes and it's up to the user to provide more by manipulating the node lists. Of course there is still basic ligature building and kerning available but one can bypass that with other code.

In MkIV, when we deal with Type1 fonts, we try to get away from traditional tfm files and use afm files instead (indeed, we parse them using `lpeg`). The fonts are mapped onto Unicode. Awaiting extensions of math we only use tfm files for math fonts. Of course OpenType fonts are dealt with and this is where we find most Lua code in MkIV: implementing features. Much of that is a grey area but as part of the Oriental TEX project we're forced to deal with complex feature support, so that provides a good test bed as well as some pressure for getting it done. Of course there is always the question to what extent we should follow the (maybe faulty) other programs that deal with font features. We're lucky that the Latin Modern and TEX Gyre projects provide real fonts as well as room for discussion and exploring these grey areas.

In parallel to writing this, I made a tracing feature for Oriental TEXer Idris so that he could trace what happened with the Arabic fonts that he is making. This was relatively easy because already in an early stage of MkIV some debugging mechanisms were built. One of its nice features is that on an error, or when one traces something, the results will be shown in a web browser. Unfortunately I have not enough time to explore such aspects in more detail, but at least it demonstrates that we can change some aspects of the traditional interaction with TEX in more radical ways.

Many users may be aware of the existence of so-called virtual fonts, if only because it can be a cause of problems (related to map files and such). Virtual fonts have a lot of potential but because they were related to TEX's own font data format they never got very popular. In LuaTEX we can make virtual fonts at runtime. In MkIV for instance we have a feature (we provide features beyond what OpenType does) that completes a font by composing missing glyphs on the fly. More of this trickery can be expected as soon as we have time and reason to implement it.

In pdfTEX we have a couple of font related goodies, like character expansion (inspired by Hermann Zapf) and character protruding. There are a few more but these had limitations and were suboptimal and therefore have been removed from LuaTEX. After all, they can be implemented more robustly in Lua. The two mentioned extensions have been (of course) kept and have been partially reimplemented so that they are now uniquely bound to fonts (instead of being common to fonts that traditional TEX shares in memory). The character related tables can be filled with Lua and this is what MkIV now does. As a result much TEX code could go away. We still use shape related vectors to set up the values, but we also use information stored in our main character database.

A likely area of change is math and not only as a result of the TEX gyre math project which will result in a bunch of Unicode compliant math fonts. Currently in MkIV the initialization already partly takes place using the character database, and so again we will end up with less TEX code. A side effect of removing encoding constraints (i.e. moving to Unicode) is that things get faster. Later this year math will be opened up.

One of the biggest impacts of opening up is the arrival of attributes. In traditional TEX only glyph nodes have an attribute, namely the font id. Now all nodes can have attributes, many of them. We use them to implement a variety of features that already were present in MkII, but used marks instead: color (of course including color spaces and transparency), inter-character spacing, character case manipulation, language dependent pre and post character spacing (for instance after colons in French), special font rendering such as outlines, and much more. An experimental application is a more advanced glue/penalty model with look-back and look-ahead as well as relative weights. This is inspired by the one good thing that xml formatting objects provide: a spacing and pagebreak model.

It does not take much imagination to see that features demanding processing of node lists come with a price: many of the callbacks that LuaTEX provides are indeed used and as a result quite some time is spent in Lua. You can add to that the time needed for handling font features, which also boils down to processing node lists. The second half of 2007 Taco and I spent much time on benchmarking and by now the interface between TEX and Lua (passing information and manipulating nodes) has been optimized quite well. Of course there's always a price for flexibility and LuaTEX will never be as fast as pdfTEX, but then, pdfTEX does not deal with OpenType and such.

We can safely conclude that the impact of LuaTEX on ConTEXt is huge and that fundamental changes take place in all key components: files, fonts, languages, graphics, MetaPost xml, verbatim and color to start with, but more will follow. Of course there are also less prominent areas where we use Lua based approaches: handling url's, conversions, alternative math input to mention a few. Sometime in 2009 we expect to start working on more fundamental typesetting related issues.

Roadmap

On the LuaTEX website `www.luatex.org` you can find a roadmap. This roadmap is just an indication of what happened and will happen and it will be updated when we feel the need. Here is a summary.

- merging engines
  Merge some of the Aleph codebase into pdfTEX (which already has ε-TEX) so that LuaTEX in dvi mode behaves like Aleph, and in pdf mode like pdfTEX. There will be Lua callbacks for file searching. This stage is mostly finished.
- OpenType fonts
  Provide pdf output for Aleph bidirectional functionality and add support for OpenType fonts. Allow Lua scripts to control all aspects of font loading, font definition and manipulation. Most of this is finished.

- tokenizing and node lists
  Use Lua callbacks for various internals, complete access to tokenizer and provide access to node lists at moments that make sense. This stage is completed.
- paragraph building
  Provide control over various aspects of paragraph building (hyphenation, kerning, ligature building), dynamic loading loading of hyphenation patterns. Apart from some small details these objectives are met.
- MetaPost (MPlib)
  Incorporate a MetaPost library and investigate options for runtime font generation and manipulation. This activity is on schedule and integration will take place before summer 2008.
- image handling
  Image identification and loading in Lua including scaling and object management. This is nicely on schedule, the first version of the image library showed up in the 0.22 beta and some more features are planned.
- special features
  Cleaning up of hz optimization and protruding and getting rid of remaining global font properties. This includes some cleanup of the backend. Most of this stage is finished.
- page building
  Control over page building and access to internals that matter. Access to inserts. This is on the agenda for late 2008.
- TEX primitives
  Access to and control over most TEX primitives (and related mechanisms) as well as all registers. Especially box handling has to be reinvented. This is an ongoing effort.
- pdf backend
  Open up most backend related features, like annotations and object management. The first code will show up at the end of 2008.
- math
  Open up the math engine parallel to the development of the TEX Gyre math fonts. Work on this will start during 2008 and we hope that it will be finished by early 2009.
- cweb
  Convert the TEX Pascal source into cweb and start using Lua as glue language for components. This will be tested on MPlib first. This is on the long term agenda, so maybe around 2010 you will see the first signs.

In addition to the mentioned functionality we have a couple of ideas that we will implement along the road. The first formal beta was released at tug 2007 in San Diego (usa). The first formal release will be at tug 2008 in Cork (Ireland). The production version will be released at EuroTEX in the Netherlands (2009).

Eventually LuaTEX will be the successor to pdfTEX (informally we talk of pdfTEX version 2). It can already be used as a drop-in for Aleph (the stable variant of Omega). It provides a scripting engine without the need to install a specific scripting environment. These factors are among the reasons why distributors have added the binaries to the collections. Norbert Preining maintains the linux packages, Akira Kakuto provides Windows binaries as part of his distribution, Arthur Reutenauer takes care of MacOSX and Christian Schenk recently added LuaTEX to MikTEX. The LuaTEX and MPlib projects are hosted at Supelec by Fabrice Popineau (one of our technical consultants). And with Karl Berry being one of our motivating supporters, you can be sure that the binaries will end up someplace in TEXLive this year.

Hans Hagen

DHZ Boek

Abstract
Het wordt steeds makkelijker om zelf een boek te produceren terwijl het voor uitgevers steeds minder interessant wordt om in nieuwe auteurs te investeren. Zelf publiceren ligt voor de hand maar als je het mooi wil doen is het een uitdaging.

Keywords
Doe Het Zelf, Boek, Online drukker, POD

Jaren geleden werkte ik free-lance voor een kleine Arnhemse uitgeverij. Voor ongeveer vijfhonderd gulden per stuk maakte ik boekjes drukklaar (typesetten in TEX en het controleren op tikfouten) en een enkele keer stelde ik zelf een klein boek met verhalen samen. De lay-out was vrij simpel en soms maakte ik gebruik van slimme vondsten voor bijvoorbeeld paginering (turftekens en dobbelstenen in plaats van cijfers) die een vriend voor me in TEX had uitgedokterd. Later verzorgde ik met de hulp van Willi Egger het typesetten van een bundel essays waarvan ik het auteursrecht heb en dat bij een 'grote' uitgeverij uitverkocht was geraakt. Op deze manier bleef het toch beschikbaar.

Al die tijd had ik niets te maken met het drukproces zelf, het ontwerp van de omslag, het aanvragen van een ISBN-nummer en de distributie via het Centraal Boekhuis. Publiciteit regelde ik in een enkel geval wel zelf maar ik had er geen idee van wat zo'n boek nu eigenlijk kost om het te produceren en wat het oplevert. Ik wist dat de Arnhemse uitgever zijn kosten zo afstemde dat hij de kosten grotendeels kon dekken met een minimale verkoop, bijvoorbeeld van 200 bibliotheken die het boek uit een catalogus konden bestellen.

Toen de essaybundel in een andere samenstelling opnieuw in druk zou verschijnen bij de grote uitgeverij, wilde ik graag dat het er mooi uit zou zien, dus ik ging akkoord op voorwaarde dat Willi Egger zou worden ingehuurd voor het drukklaar maken. Hier bleek dat ook een grote uitgeverij aan een Amsterdamse gracht op de kleintjes let, want veel geld was er niet beschikbaar, men wilde graag een register maar er was eigenlijk geen budget voor een literair-historisch deskundige die zou aangeven welk notenapparaat er moest komen en het bindwerk was uiteindelijk ook niet precies wat je zou verwachten bij een uitgave met de titel "Het voorbeeldige boek." Willi heeft later in zijn handbinderij nog wel voor ons ieder een exemplaar gebonden, chique, in een foudraal.

Intussen heb ik materiaal verzameld voor een nieuw boek met tekst en foto's. Zoals David Walden in zijn artikel schrijft, kun je energie stoppen in het vinden van een uitgever die zijn know-how en zijn enthousiasme wil inzetten om die dingen te laten doen die ik nu nog niet zelf kan, maar je kunt ook gaan leren hoe je de nodige stappen zelf zet.

Er kan nog een argument zijn om het zelf te doen. Dat boek van mij is misschien maar voor een handvol mensen de moeite waard en dan is het pas interessant voor een uitgever als die het erg leuk vindt én zich de verliezen kan veroorloven door kaskrakers in het fonds.

In *Het Parool* van 16 april 2008 vertelt uitgever Wouter van Oorschot hoe het gaat als je lang durft te wachten tot het werk van een auteur gaat lopen: "Dat soort uitgaven doe je door interne subsidiëring: je zet een deel van je winst op bestsellers weg voor de magere jaren, het andere deel investeer je in essays, verhalenbundels, poëze en debutanten, zoals je weet niet de best renderende genres, om eens een understatement te gebruiken." Dan zegt hij het nog voorzichtig.

Een uitgever is, zo blijkt uit een staatje bij het interview, al snel 1000 Euro kwijt aan het nawoord, 1000 Euro aan ontwerpkosten, 2000 Euro voor het zetwerk. Vierduizend exemplaren drukken kost 3000 Euro voor de drukker, een kleine 6000 Euro aan papier en ruim 13.000 Euro voor de binder. Verdient dat lekker? Nou nee. Als het boek 34 Euro kost in de winkel dan houdt de uitgever daar 1,29 Euro aan over. Dan moeten er wel bijna vierduizend exemplaren worden verkocht, binnen twee jaar. Op die magere winst moet de uitgever dus ook nogeens jaren wachten. Het zou beter zijn als het boek 58 Euro mocht kosten, maar dat vinden de boekenliefhebbers te duur, legt Van Oorschot uit: "Het boek staat laag op de hitparade van niet-primaire levensbehoeften. Eerst gaat men naar het café. Dan koopt men audio-apparatuur, fototoestellen, mobieltjes, iPods. En eens per jaar een nieuwe iPod, omdat de vorige verouderd is. Dan gaat men nog eens eten in een goed restaurant. En na nog van alles en nog wat, dan pas overweegt men een boek te kopen. Want boeken zijn dus o zo duur…"

Het wordt duidelijk welk risico een uitgever neemt als hij boeken echt mooi wil maken en hoe praktisch het is voor andere uitgevers om het simpel te houden en boeken te maken die vlot verkopen en die goedkoop kunnen worden geproduceerd voor lezers die het niet zoveel uitmaakt hoe het eruit ziet.

Op internet zijn er bedrijven zoals www.blurb.com en myphotobook.nl waar je via een web-interface je foto's en teksten simpelweg het boek binnensleept. Een bevriende professioneel fotograaf (www.hans-franz.nl) met een scherp oog voor afdrukkwaliteit heeft er wat ervaring mee opgedaan. De productiekosten van zo'n boek kunnen gemakkelijk oplopen tot 50 Euro per stuk, wat niet duur hoeft te zijn. Als de fotograaf voor een enkele foto van een opdrachtgever meer krijgt dan de lezer voor een heel fotoboek betaalt, dan is dat boek niet echt duur te noemen. Het is wel van belang een goeie producent te kiezen omdat de kwaliteit van het fotografische drukwerk wisselt.

Een ISBN-nummer is wel van belang als het boek bestelbaar moet zijn bij de boekhandel om de hoek of online. Bij portal5.boekhuis.nl kan een ISBN-nummer worden aangevraagd.

Als oude TEX gebruiker ben ik intussen wel verwend met nette typografie en ook met de ervaring dat ik de vormgeving in eigen hand heb. Ik deins ervoor terug om teksten door een luik op internet een boek in te werpen.

Nu kan ik twee kanten op: met TEX een PDF maken en een drukker zoeken die daar iets van kan maken met dezelfde prijs-kwaliteitsverhouding als het werk van bijvoorbeeld myphotobook.nl, of de met TEX opgemaakte pagina's stuk voor stuk exporteren en in hun vormgeving via Photoshop converteren naar TIFF of JPG voor de web-interface van de drukker.

De komende tijd ga ik hiermee aan de slag. We zullen zien.

Frans Goddijn
frans@goddijn.com

Notes on Self-publishing[*]

Abstract
This note summarizes what I have learned about self-publishing.

Keywords
self-publishing, publishers, cost

Over the past couple of years I have gotten pretty deeply into the world of self-publishing. Even before that I drafted a book using a professional typesetting system, although the traditional publisher of that book retypeset it for actual publication.[1]

The purpose of this note is to summarize what I have learned and some thoughts I have about self-publishing. I originally drafted these notes in mid-2007, and I believe what I say here was accurate then. However, technology and publishing economics are changing rapidly; also I am not a publishing expert. Thus, after reading these notes, the reader should do enough additional research to form his or her own views about the validity of the various issues I raise relating to self-publishing.

Publisher options

Communications and transportation technology is leading to disintermediation (elimination of middle men) in many fields. Publishing is no exception. Let's look at the options an author has.

A traditional publisher

If you are an author and you want massive PR and bookstore distribution for your sales, then you need a traditional publisher.[2] Except, they won't give you massive PR unless they think your book is going to be wildly popular or you are already a wildly popular author. You should also seek a traditional publisher if you want someone else to foot the development bill (editing, illustration, permission, design and layout, indexing, printing, and perhaps an advance for you). Except, it is hard to get a traditional publisher interested in you (you probably need an agent or personal contact). (Of course, the amount of effort to self publish is considerable; if your book has a decent chance of being popular enough making a substantial profit, putting that same amount of effort into finding a publisher rather than into self-publishing may well result in you finding a publisher.)

If you succeed in getting a traditional publisher, the publisher is going to want you to sign a contract that gives it the worldwide rights in all media, and your work may well be tied up so you have to ask the publisher's permission to reuse a chapter elsewhere. For books on specialized topics (e.g., scientific, medical, technical, or professional), the publisher may ask you to promise to buy some minimum number of copies (e.g., 1,000) to cover their costs.[3] If the book is no longer selling many copies (i.e., is essentially out of print), you will have to ask the publisher to revert the rights to you, and there is a possibility that the publisher will refuse. With the possibility of print-on-demand or POD (where one can order single copies of a book to be digitally printed[4]), I suspect publishers increasingly will use POD to keep books technically in print, selling a few copies per year in the later years when doing large lithographic print runs no longer makes sense.[5]

Working with a traditional publisher, you may get 10 percent of the list price after returns (in the publishing industry in the United States it is apparently traditional that book stores can send back for full credit all the books they ordered even if they are damaged). With certain types of books, the author's percentage may be after returns and on the wholesale price (i.e., 45 percent of list price) with a proportional decrease in the return to the author.[6]

A subsidy publisher

One alternative to a traditional publisher is "vanity" or subsidized publishing where they make their money by having you pay for their book development services, you get a few books, and that's the end of it; they typically don't make their money by selling lots of copies of your book. They may have a website where your book is sold, but you won't get a very big share of the price. Also, they are the publisher of record, which ties you up in various ways. With this option the odds are against you making enough money to justify having given up control. Nonetheless, it may be the right option for someone who just wants to create a few copies of a book to give to family members and friends.[7]

[*]Copyright Dave Walden © 2007, 2008

Self-publishing

Another alternative to a traditional publisher is self-publishing. People have always done self-publishing when they thought they had something to say and could not interest a regular publisher or when they simply wanted a few copies printed and didn't want to get involved with a subsidy publisher. Googling for "famous books that were self published" will show several links to famous books that were originally (and perhaps permanently) self-published.[8]

In self-publishing you acquire the book's ISBN number yourself, and you do the development work yourself or hire someone to do the various parts for you. You control how the book is printed and by whom, the distribution, and what limited rights you give to other people as it is beneficial to you. Self-publishing can be an excellent option in appropriate situations.

The rest of this article is about self-publishing.

Ease of printing in the digital age — an example

Initial publication

In the fall of 2006, I self-published the book *Breakthrough Management* (by Shoji Shiba and me — www.walden-family.com/breakthrough). I chose to self-publish primarily because I wanted to experiment with the breakthrough technology of self-publishing and digital printing; this seemed appropriate given the topic of our book. Also, my approach to authorship, even for a traditional publisher, involves designing the book and typesetting it for submission to the publisher.[9] Thus, I did not anticipate a lot more work if I went the route of self-publishing. My co-author and I also had an existing, highly targeted market — people who already know my co-author and me and our previous writings, or who are introduced to my co-author at one of the management classes he teaches throughout the world.

In general, I was pleasantly surprised with how easy it was to have a book printed that was prepared in LaTEX.[10]

I sent the PDF output of LaTEX to the printer in Delhi chosen by our publisher in India (where my co-author was doing much teaching and which provided the ISBN number for the book), and the book was printed without any additional interaction by me except to check a printer's proof (which was fine). The Indian print run was 1,000 copies using lithographic printing because, purportedly, print-on-demand was not available in India.

(Again, see the first paragraph of the section "Printing options" on page 56 for the distinction between offset and digital printing.)

In the United States, I wanted to use print-on-demand where I would not have to invest in and find inventory space for a print run of 1,000 copies. I requested by email quotes from eight printers who advertised themselves as POD printers, using the general quote format suggested in Pete Masterson's book, *Book Design and Production: A Guide for Authors and Publishers* that is aimed at small publishers and self-publishers. I received several plausible quotes by return email.

One of the geographically closest printers (Ames On Demand of Somerville, Massachusetts, about 60 miles from my home on Cape Cod) also had nearly the best price. I phoned him and asked him two questions: (1) I told him I already had a ready-to-print PDF and wondered if I would have to make any adjustments to my page layout (e.g., text block size) for him to print the book on his presses; (2) Could he send me an example of a photographic image that had been printed on his presses (Xerox 6180 for the black and white text and iGen3 for the color cover) so I could review the reproduction quality of an image.

The printer suggested that I send him my PDF file and he would send me back a proof. I was thinking he would print *a page* with a photographic image and send it to me. I was happily surprised when the overnight delivery truck arrived two days later with a finished, bound proof of the whole book including my cover art.[11] Obviously this is an advantage of digital printing. Since finished copies are being printed a page at a time rather than on large offset sheets with multiple different pages on each sheet, it is easy for the printer to run a complete copy of a book through his digital printer (just like he would do successively for hundreds of copies).

The proof sent by the printer looked great. I asked him to slightly shift the title on the book spine (I sent him an adjusted copy of the cover art), to slightly shift the text block on the page to increase the inside margins and decrease the outside margins (he was able to do this without me touching anything in LaTEX), to bind the book using a matte rather than glossy coating, and to make a one line change to the back-of-the-title page (I provided a single new PDF page which the printer used to replace the previous page in my whole-book PDF file). He did these things, a whole-finished-book proof arrived at my home a few days later, and I gave him the go ahead to print 250 copies of the book (his quoted price was the same per book for volumes of 250, 500 and 1,000 books, so it was an easy decision to print the minimum of the quoted number of copies).[12] A few more days later, I picked up the printed copies at the printer's loading dock with my little pickup truck (I did the pickup myself to save shipping costs).

The printer said he would keep my PDF files as he had adjusted them and could print more copies at any time.

Reprinting

By the spring of 2007, the *Breakthrough Management* book print runs of 1,000 copies in India and 250 copies in the United States had been sold out. Also, a number of typographical errors in the book had been found by that time. Therefore, I updated the LaTeX source files for the book (which was possible with almost no changes in pagination), and the Indian publisher obtained a new 13-digit ISBN number for the book (as of January 1, 2007, the world shifted from 10-digit to 13-digit ISBN numbers).

With the new 13-digit ISBN number in hand, in late June of 2007 I updated the ISBN number and bar code on the back cover part of the cover art work, and I sent the new cover file and interior file to India for reprinting there. Simultaneously, I reformatted the page sizes of the cover and interior slightly to meet the requirements of Lightning Source Inc. (LSI), and sent the files to them to print which in turn facilitates the books being sold via Amazon and the other Internet-based book stores.

(**NB:** In the spring of 2008, there has been much consternation among authors and self-publishers about Amazon forcing use of its own print-on-demand company, BookSurge; the web page www.writersweekly.com/amazon.php is dedicated to this issue. I suppose that authors about to publish a new book can avoid any problem of using LSI by just using BookSurge for their printing, although there is some discussion of it being more expensive. People already using LSI for a book may have a bigger problem, although presumably other on-line bookstores will continue to sell books printed by LSI. As of April 30, 2008, Amazon was still selling my LSI-printed book.)

The printing from LSI was slightly less sharp than that from Ames, and the photographic images were a bit muddy in comparison to the Ames printing. Nonetheless, this provides a useful alternative path for people to buy the book (from the on-line book stores), and I can order copies myself at the cost of printing plus shipping to me or drop shipping to my customers.

I also asked LSI to make my book available through their UK branch so that I can order books printed and shipped from there if that will reduce the cost to get books to European customers. This worked smoothly without me having to pay anything more or upload my files again. Within a day, the book was listed with www.amazon.co.uk, and a few days later the book was also shown on the Amazon UK website as being for sale from other book stores which were undercutting the Amazon UK pricing for the book.

Another couple of weeks later, the book was also shown for sale at www.amazon.com in the United States. It is also listed for sale by Amazon in other countries (Austria, Canada, France, Germany, and Japan) and by other book stores with an on-line presence such as Barnes & Noble and Powell's Books.

In parallel with setting up to work with LSI, I sent the new cover and interior files to Ames On Demand where I continue to order books a carton (e.g., 25 or 30 books) at a time and mostly use these for filling bulk orders that come from conferences in which my co-author participates or companies at which he consults.

Financial details

I am not really involved in printing and selling our book in India, except to give the files to the organization there for which my co-author consults.

The development cost of my 280 page book was \$1,450 for editing (a fixed price estimate based on the editor looking over my draft manuscript), \$1,310 for illustration (at a hourly rate), \$575 for proofreading (at an hourly rate), and \$150 for help setting up to use the Minion Pro fonts. (I did the design and layout for my current book myself because I wanted to learn how to do it and wanted to experience "going all the way" with LaTeX). All of those costs seemed pretty reasonable to me, given the quality of service I received. I also paid \$50 for permissions (other permissions were obtained gratis). In total my development cost was about \$3,800 (including some phone, fax, and postage costs but excluding my time).

You can perhaps extrapolate from the figures in the previous paragraph to possible costs for the various functions for a book of a different size. My book had well over a hundred, often complicated, line drawings; yours may have none. My book also was highly technical with lots of references, etc., requiring lots of editing; your book may be simpler and thus less time consuming to edit. A simpler book would also be easier to typeset.

One can get the ISBN numbers for approximately \$300 for a block of 10 in the United States — see www.www.isbn.org.[13]

I already have a website, and it costs nothing more to sell my book from it. For most books I sell from my website, I get copies of the book in lots of 20 or more from Ames On Demand for \$5.68 per book plus a 5 percent Massachusetts sales tax[14] (they charge no setup fees). This was a good price compared with other quotes I received, and I liked their quality and the fact that they are located near me.

When I sell the book via my website, www.walden-family.com/breakthrough, I quote a price that includes the list price of \$30 and part of the shipping cost (e.g., a total price of \$32.50). Buyers click on an appropriate link to PayPal on my website and pay me either using PayPal or PayPal's capability to accept credit card payments. In these instances I typically absorb $2.50 (or a few dollars more) of the shipping costs. PayPal also deducts their fee (something like $1.50 to collect $32.50). I don't attribute packing costs to each book, but perhaps such consumables are another dollar per book leaving a net profit of perhaps $20 per book after printing, shipping, and collection costs. At that rate I made back my development costs after selling about 200 books.

In those cases where a company or conference orders a dozen or a few dozen books at time, I have been giving significant discounts, for instance, selling the books for $10 each and charging $5 each for shipping. Even at this very discounted price, I still clear about $4 per book (not counting my labor).

These bulk orders are the most work for me. The buying organization often wants me to provide an invoice and wants to pay via an inter-bank transfer. I have a prepared invoice form in Excel which I modify appropriately for each order and send as a PDF email attachment. The invoice specifies that the buyer must pay all currency exchange, bank and customs fees and will send me my price in US dollars exclusive of those other items. The only additional cost I absorb is the $10 fee my bank charges to receive a wire transfer (and sometimes I bury that fee in my quote). I have a separate bank account to which wire transfers come and I sweep money out of that account and into another bank account as soon as the money arrives (I am a little afraid of leaving a substantial amount of money in a bank account that can be accessed for a wire transfer). I must pack the books especially well for shipment in heavy boxes (and in one case the books were returned to me and I had to repack and reship them because of a customs mixup in the destination country). I must fill out a larger export form at my post office. And sometimes there is confusion at the sending bank about how to wire money to my bank account, and I have to repeat my instructions to resolve the confusion. (Probably I would be justified in not discounting the book price so much for such orders.)

I also arranged for the corrected printing of the book to be available through Lightning Source in the US and in the UK. They charged setup fees of about $130 ($50 for me to submit my cover, $42 for me to submit the text of the book, $30 to send me a proof via overnight shipping, and some small fee for a catalog listing). For them, I priced the book at $30, 24 Euros, and 17 GBP[15] with a 55 percent discount. Thus, for books sold via book stores (e.g., Amazon), I get about $13.50 per book regardless of what price they sell the book for (I also specified no returns), and LSI subtracts the printing cost (about $4.55 per book) and sends me the rest. I also am not involved in fulfilling these orders in any way. The book store already has collected the cost of shipping and passed it to LSI. My only involvement is to note the money arriving some weeks later in my PayPal account and then transferring it to my bank account. Many buyers prefer to buy from a real company like Amazon rather than an individual's website.

When I sell a book via my website but have the printing done via Lightning Source, I pay about $5.10 per book, plus a $1.50 handling fee per order, plus the price of them drop shipping the book to the customer (which appears to be more expensive than when I shop books myself except intra-Europe). Thus, my net is still probably about $15 per book. In this case, I have to login to the Lightning Source website, provide the shipping address and number of books, and provide my credit card information for what they charge me; doing this can take 5 or 10 minutes per order.

My bookkeeping is simple. I keep track of income (minus financial fees), cost of goods sold (printing and shipping books), and development costs in any given year. For tax purposes, the net of the income and costs either adds to or subtracts from the rest of my personal income for the year.

Deal making

In the summer of 2007, I also began talking with a group in Hungary about providing them translation rights to the book, for which I still plan to act as publisher in name but with them paying for the translation, retypesetting, and printing, and doing local selling. I am also talking to a couple of people in Spain about publishing a Spanish translation. I don't know if anything will really come of these discussions, but being my own publisher gives me lots of deal-making flexibility.

I also am considering an ebook edition of *Breakthrough Management*, a decision I can make alone.[16]

Other issues relating to self-publishing

Printing options

There are several printing options: *(a)* traditional lithography or offset printing where a sequence of pages (for instance, 16 pages) is printed on a large sheet of paper and the pages are cut apart and put in order later in the process — this is the most economical method of printing (in terms of cost per book) with print runs of perhaps a thousand copies or more, but it also may be necessary for shorter runs of books that have lots of color or non-line-art that you want to print well (line art works fine without lithographic printing); *(b)* short run lithography which is not economical on a per book basis but may be the way to go if you need the quality even though you may think you cannot sell more than a few hundred books; *(c)* digital printing (e.g., sort of like your home laser printer but faster) which can be done very well or not quite as well depending on the care that is taken (I have seen very fine black and white photos done with digital printing). Print-on-demand (POD), where you can buy one to hundreds of copies at a time, typically uses digital printing, and provides the major advantage that you don't have to pay up front for a thousand or more books to be printed and then store them until they are sold.

There are POD printers, e.g., lightningsource.com and lulu.com,[17] which will take your PDF file of the text of your book and the PDF of your cover, charge you a modest setup fee, store your book electronically on their computers, and print one or many copies for you at a relatively fixed price per copy, e.g., $6, whenever you order them. If I had a book without photographic or fine art images (e.g., a management book or a novel), this would be a good option.[18]

Books printed by Lightning Source Inc. (LSI) also reach wholesaler catalogs (particularly that of Ingram which is a sister or parent company of LSI) such that anyone can order the book from the wholesaler.[19] You set the list price and the discounted price, and LSI fills orders as they come in, e.g., from a retailer such as Amazon,[20] and basically sends you the difference between your discounted sales price and their printing cost. If you list the book at $30 and discount it to a wholesale price of 50 percent, they will send you the difference between $15 and the price they charge you to print a copy for each one that is sold, which will probably net you something like $10. Note that Amazon, etc., will now list your book but the "brick and mortar" book stores still will not carry it except to order it when a customer prepays because they insist on being able to return books they order for their book shelves. If only few people order your book from Amazon, Amazon will show a multi-day shipping period which may discourage buyers, but if your book starts to sell well and you offer a wholesale discount Amazon thinks is appropriate (e.g., 55 percent), then Amazon may begin to inventory it and list a shortened delivery time.

The other alternative is just a regular printer (big or small) who gives you the best terms when you ask a few for quotes, but that will not connect you with Amazon without additional steps.

Distribution

I use the word "distribution" in this paragraph in an informal sense, not in the publishing jargon sense. (In publishing, a distributor is a business that typically has an exclusive contract with the publisher, i.e., you, to find places that will sell your book.)

In the case where you use a regular printer, you can sell the book yourself (e.g., as I do via my website), you can consign it to someone else (e.g., your local historical society to sell your book on local history), or you can join Amazon's Advantage program where you ship the books to them for inventory and they list and sell the books and take a commission. In another Amazon program (Marketplace), you inventory the book, they list it, they sell it and send you the order, you fulfill the order, and Amazon sends your share of what the customer paid (e.g., minus their sales commission). (You can also hire order fulfillment houses who will charge you $3 or $4 per book, i.e., you initially send the fulfillment house an inventory of your books, you take in the money when you sell a book, and you send them the order plus their fee plus postage for them to mail the book to your customer.)

You also can do combinations of the above. For instance, as described in the section "Ease of printing..." on page 54, I have Lightning Source print the book which will be wholesaled by Ingram (the largest commercial wholesaler in the United States) or Baker and Taylor (the largest library wholesaler) and simultaneously have the book printed by another (perhaps higher quality) printer or buy copies from Lightning Source and sell them myself in one way or another.

Lightning Source's branch in the UK is connected with the big European wholesalers such as The Bertram Group (www.bertrams.com) and Gardner's Book Service (www.gbsbooks.com). Amazon also has branches in other countries and their Amazon Marketplace (where they list the book but you fulfill the order) also permits you to sell and ship overseas.

Finally, since you control the book when you self-publish, you have flexibility to deal with multiple entities to "publish" the book rather than all rights being tied up with a single traditional publisher, although using multiple "publishing" paths may well not be the best overall marketing strategy. Nonetheless, you could let your local historical society be the "publisher" for your local history book, changing the title page to list them as publisher and having them get their own ISBN number which you will put in your electronic file, selling them copies wholesale, and letting them sell the book locally or to people who order it from them. Simultaneously, you could publish the book yourself (using your own ISBN number) for web-based sales to whomever orders the book from you.

Disintermediation and flexibility

The point of all this is that once you decide the traditional publisher is not for you and you prepare a ready-to-print file for your book, then, in return for your initial investment, you can control everything and make whatever deals you want, and modern printing and distribution technology offers many options you can use.

Of course, there are advantages in many situations to working with a traditional publisher. The advantages include access to the publisher's editors, indexers, typesetters, art department and established distribution channels, the publisher's payment of the development costs of the book, and not going against the established model for how a book gets published. In my view, the major disadvantages of going with a traditional publisher are *(a)* the publisher ties up all the rights and the author loses control of his intellectual property, and *(b)* it is often hard to get a traditional publisher to take you on and let you produce the book you want to produce.

In any case, my experience has made it clear to me that it is now entirely feasible and relatively inexpensive, in cases where one is willing to forego the advantages of a traditional publisher, to self-publish. TEX, LaTEX, ConTEXt, etc., are available for free for anyone who already knows one of them or is interested in learning to use one. (The visually oriented typesetting program Scribus is also free.) PDFs (easily output by all typesetting systems) appear to be a nearly universal way of transmitting ready-to-print manuscripts to a printer; even when the printer uses traditional offset printing with many pages per large sheet of paper, in my experience the printer takes care of whatever is required to turn a sequence of pages in a PDF file into many pages on a sheet. And capabilities like PayPal, Amazon, rapid international shipping, and website-based selling make it possible to sell a book world wide.

Marketing, promotion, and good business

Of course, practically speaking, you still have to promote the book in order to get people to want to buy it.

Marketing and promotion involve all of the things that happen with traditional publishers. If you hope to have the book reviewed, you must send out review copies, typically in advance of publication. Parallel short publications and presentations can help sales. Sending out some sort of notice to targeting mailing lists might help. Some authors have blogs that augment the content of their books. A YouTube video to go along with the book is now possible. Having good positive comments about your book on Amazon.com should help. Having great success with a prior book undoubtedly helps. And so on. Dan Poynter's book (see the next section) has good content on promotion of self-published books.

A nice looking cover can also help sales. Yuri Robbers has written an article (tug.org/pracjourn/2007-1/robbers) that discusses some of the issues of book cover design (you can ignore his discussion of how to use the PStricks system to implement your book cover design). Morris Rosenthal's book (see next subsection) recommends a simple cover that stands out in the small size that will be displayed by the on-line book stores.

Probably you are no more likely to make money self-publishing a book than you are to make money with a book published by a traditional publisher. Most books do not make a lot of money and most authors are not rich. However, since you may be putting up your own funds to develop and initially print a self-published book and may incur other liabilities, you do have to think about it in a businesslike way. You have to understand the difference between fixed and variable costs and the break even point. You probably don't want to invest in a big printing run before you have strong evidence you are going to sell a lot of copies. If you expect to sell a material number of books at a profit, you need to learn how normal businesses operate (e.g., approaches to limiting liability, registering as a business in your state, collecting state sales tax, etc.) so you don't accidentally get into trouble.

If your book does become popular, you may make more money publishing it yourself, and many people apparently do make money self-publishing. Also, by self-publishing and retaining all the rights, you are free to later make a deal with a traditional mainstream publisher, if one becomes interested after seeing the popularity of you book.

Resources for learning more about self-publishing

While I think I have presented some relatively accurate information about self-publishing, I am no expert. Here are a few useful books by experts on self-publishing:[21]

- ☐ Dan Poynter's book *The Self-Publishing Manual* has gone through many revisions; he also has a useful website: www.parapublishing.com.
- ☐ Pete Masterson's book (mentioned above) on *Book Design and Production, A Guide for Authors and Publishers* is more about the nuts and bolts of publishing (not the writing and promotion like Poynter's book); he also has a useful website (www.aeonix.com) including lots of useful lists, e.g., printers, book coaches, small publishers, etc.
- ☐ Morris Rosenthal has a good small book on Print-on-Demand Book Publshing; his website (www.fonerbooks.com/contact.htm) also include a reprint of his book's text on the economics of POD publishing versus traditional publishing.
- ☐ Robert Bowie Johnson and Ron Pramschufer's

book *Publishing Basics* is a good, short, very basic introduction to things. Pramschufer also has a related business and useful website (where you may be able to get this short book for free): www.selfpublishing.com.
- □ Aaron Shepard's book *Aiming at Amazon* (available from amazon.com) will be valuable reading for anyone hoping to sell his or her book via Amazon and similar companies.
- □ Another interesting website is www.gropenassoc.com — click on "Reference Desk"; this is run by one of the three people who moderates the Yahoo Self-Publishing discussion group. Among other things, she recommends several books on the business.

You can also hire people to help you with any of the aspects of self-publishing including hiring a "book coach" to guide you through the process.

I think one of the best things to do to understand self-publishing is to subscribe to the Yahoo Self-publishing discussion group and read the messages that go by for a few weeks. (They also have an online archive of past messages.) All of the people whose books I just listed and many other very knowledgeable people participate in this list. When it comes to discussing typesetting programs, the Yahoo Self-publishing discussions focus on the commercial typesetting and layout systems, e.g., InDesign and QuarkXPress. The free Scribus system is also mentioned from time to time. Use of Microsoft Word for typesetting and layout is frequently denigrated in this discussion group, although Aaron Shepard has a book (*Perfect Pages*, available from Amazon) on how to use Word for these functions. The just mentioned systems all use a graphical user interface.

I use LaTEX, based on TEX, as my typesetting system. However, LaTEX uses a fundamentally different paradigm (non-graphical) for specifying how the typesetting will be done. Consequently, it is not a popular topic of discussion within the Yahoo Self-publishing group, although its use is regularly encouraged by one of the group's three moderators (John Culleton[22]); John himself does not use LaTEX, which he finds too confining, and instead uses TEX itself and ConTEXt which is also based on TEX.

Hiring people versus doing it yourself

Even though you are self-publishing a book, you can hire some or all of the tasks done for you. What you spend your own time learning to do and doing and what you hire someone to do for you is always a tradeoff. However, I would not automatically assume that a "professional" will always be able to do a better job than you.

"Professionals" in many fields come with a distribution of skill levels. Being a professional alone is not a guarantee of skill level; being a professional may only mean someone spends full time working in the area or makes a living from the area. Some professionals are very expert, most are competent, and some apparently are not really very good (for instance, I regularly see books that seem badly designed or badly typeset, including from big and well known "real" publishers[23]). A competent professional typically can handle many different situations in an efficient manner; they also have developed an eye for seeing the issues. However, competent professionals did not get that way overnight. They surely gathered the skill to deal with a variety of situations in an efficient manner over years of implicit or explicit study and actually working on a large number of different projects. They learned new things with new projects. We can do the same thing. And once we amateurs have gone past complete novice-hood in one of the aspects of book publishing, we may be as good as at least some of the professionals. In any case, amateur's level of skill may be sufficient for the book the self-publisher wants to produce, versus the money the self-publisher want to spend for professionals.

Having said that, I think some functions are probably harder to get "good enough" at. The more "crafty" skills (e.g., typesetting[24]) are probably easier; the more "arty" (e.g., book design) are probably harder. The path to becoming an excellent editor is certainly a long one, while the path to becoming good at doing line drawings with a software package is relatively short.

Typesetting, as one example

Let's look at the particular question of whether one should hire someone to do a self-published book's typesetting or should learn to do it oneself.

First, it seems to me that the issue of book design and typesetting of the book is often too tightly coupled when discussing this question. For the most part, I see book design and typesetting as quite separable issues, although of course they can be done by the same person.[25] I suspect most very basically designed books will mostly require quite normal typesetting skills, and the typical fiction book with just lots of text probably doesn't need more than a normal level of typesetting skill.

Thus, deciding whether to typeset a book oneself or hire a professional could involve answering the following questions:

- □ Are you already hiring a designer who perhaps offers a good rate for the complete package

including typesetting?
- ☐ Is typesetting interesting to you (or might it become interesting)?
- ☐ How much time versus money do you have?
- ☐ How complicated is your project and do you suspect you can learn enough to do a decent job with your project?

Regarding the last point above, I'll sketch how I developed a level of skill at typesetting.

- ☐ I choose a typesetting system (LaTeX) that I believed gave the most help in doing the various typesetting functions for me in a standard way. (In addition to being very powerful typesetting systems, the most powerful in some domains, the TeX-based systems are free and don't require more of a learning curve than the commercial alternatives such as InDesign or Word.)
- ☐ I then practiced using LaTeX by typesetting some letters, an article, and a book chapter.
- ☐ For my first book project, I chose a quite simple design. As in many crafts, copying someone else's approach is an efficient place to start.
- ☐ I felt confident that by selecting a sufficiently basic design, I could learn what was necessary (and I was willing to take the time to learn) to produce an adequately typeset (not embarrassing) product.
- ☐ I then drafted and incrementally revised my whole book using LaTeX. With each revision I improved the content of the book and made changes to my book design as I learned more about LaTeX.[26]
- ☐ Along the way, I also skimmed some books on the craft of typesetting.[27] Eventually, I skimmed a few other typography books and manuals and began to observe how books I bought or borrowed from the library were typeset.
- ☐ If I had to do it again, I would assume that I might not be able to see some of the issues, and I would try to find someone with an experienced eye to glance over my output and give me feedback on things that "must be fixed." I would do this with one chapter before typesetting the rest of the book. (In a complicated way, I in essence did this, but not explicitly.)

What I did myself, and didn't do

Although I have done lots of editing, I don't believe I can do a good job of editing my own work; therefore, in developing *Breakthrough Management* I hired an excellent editor with whom I had worked before. More generally, it is probably always a good idea to have someone else edit your own writing — more about editing in a later subsection.

Although I can use Illustrator to create line drawings, to save my time I hired an illustrator I had worked with before to do initial versions of my hundred or so line drawings. I did the corrections to them myself.

I know a decent amount about Photoshop and used it to adjust photographic images for printing.

I hired a proofreader who was recommended to me, as I could not do that fussy job well for a book length document. Like editing, it is probably always a good idea to have someone else do the proofreading, including perhaps someone different than the editor.

If I had had an index, I would have hired an indexer I have worked with before; indexing is a skill I don't have and am not interested in learning.

I did all the dealing with permissions, printers, etc., as I wanted to learn about these aspects of publishing.

I did my own typesetting, as I was confident I could do a good enough job and I wanted the experience of typesetting a book for publication.

I set up my own website for book sales, because I already have this skill (and strong biases about *not* using certain website publishing tools) and because I wanted to learn about selling via PayPal.

I am an experienced (albeit not professional) bookkeeper, so I do my own bookkeeping, invoicing, etc., using QuickBooks.

I designed my own book cover using Illustrator with some inputs coming via Photoshop (see the figure on page 62). As with typesetting, I felt confident (rightly or wrongly) of my ability to incrementally develop a decent cover, and I was interested in learning by doing.

In general, a cover is something for which I would seek a number of reviews from other people. (In fact, for a future book, I dream of programming my website to randomly display different cover designs, including different title options,[28] to see which one results in the most buying interest and then adopting that as the permanent cover for a new book.)

I initially did all of my own shipping, because I wanted to learn about modes and costs for shipping. More recently I have added an option for placing orders for drop shipments to customers via orders to Lightning Source.

In other words, I chose doing things myself where I had or thought I could develop the skill rather than paying to have them done; I hired things done I didn't feel I would do well or was uninterested in doing.

My original decision to self-publish the book was based on there being a built-in market for it of people who know of the work of my co-author and me, and thus I concentrated on sales to the people who already know us and did not attempt more general marketing. (As I refine these notes, it is dawning on me that I should seek a deal with the publisher of the previous books by my co-author and me (*Four Practical Revolutions in Management* and *A New American TQM*) to cross market those books and our self-published *Breakthrough Management* and another book we are writing now for self-publication.)

The writer-editor relationship

Self-publishers frequently ask for recommendations for good editors and reasonable prices to pay for editing. While these are necessary questions (of course one wants a good editor at a reasonable price), these are not a sufficient set of criteria. Of key importance is the personality and way-of-working match between the writer and the editor.

At the simplest level, does the writer want edits marked with pencil or pen on paper or does the writer want edits done electronically to the writer's source files? I have met editors who will only work on paper and editors who will only work electronically; some editors will work either way. In any case, the writer needs the editor to work in whichever way the writer prefers, unless the writer doesn't care. (Personally, I can accept editing of my electronic files if I am submitting something to a journal that insists on such editing. However, if I am paying the editor, I will find an editor that is happy to do the editing on paper.)

I have worked with three kinds of editors: (1) development editors who gave me general and overall suggestions but did not do a detailed copy edit, (2) copy editors who focus on the grammar, spelling and punctuation but don't really understand the content, and (3) editors who can understand the content and suggest improved ways and organizations for saying what I want to say as well as making the grammar, spelling, and punctuation adhere to a standard and be consistent. I have also worked with editors who (at least) gave the appearance of having respect for me as the author and my way of saying things while helping me improve my document, and I have worked with editors who seemed to be saying that I was just a technical person (I write non-fiction) who couldn't be expected to know proper grammar and punctuation. For a large project where I am paying the editor, I want the third type of editor and an editor who respects me while finding a way to tell me some things I probably don't want to hear but which are good for me.

One way to find an editor to whom you are well matched for a book length project is to try various editors on short projects (e.g., a paper for a journal). Alternatively, you could ask a well recommended editor to edit the preface and first chapter of your book (for an hourly fee), explaining that your goal is for you both to figure out if you are well matched to *each other*. Be sure to include in your practice run some follow-up discussion about some of the things the editor recommends, so you learn how the editor reacts when you don't automatically accept his or her suggestions.

Regarding a quote, the best thing, I think, is to have a complete manuscript available to show to the editor — as good a manuscript as the writer can do before seeking editing help. From a complete manuscript (including figures, captions, reference citations, etc.), the editor should be able to estimate the number of hours at an hourly rate or the total cost of the job, in terms of the absolute length of the manuscript, the complexity of the manuscript, and the quality of the writing.

Looking forward

My current assessment of self-publishing

I have found my experience of self-publishing *Breakthrough Management* to be successful enough to do it again. Among the self-publishing projects I am currently planning or executing are the following:

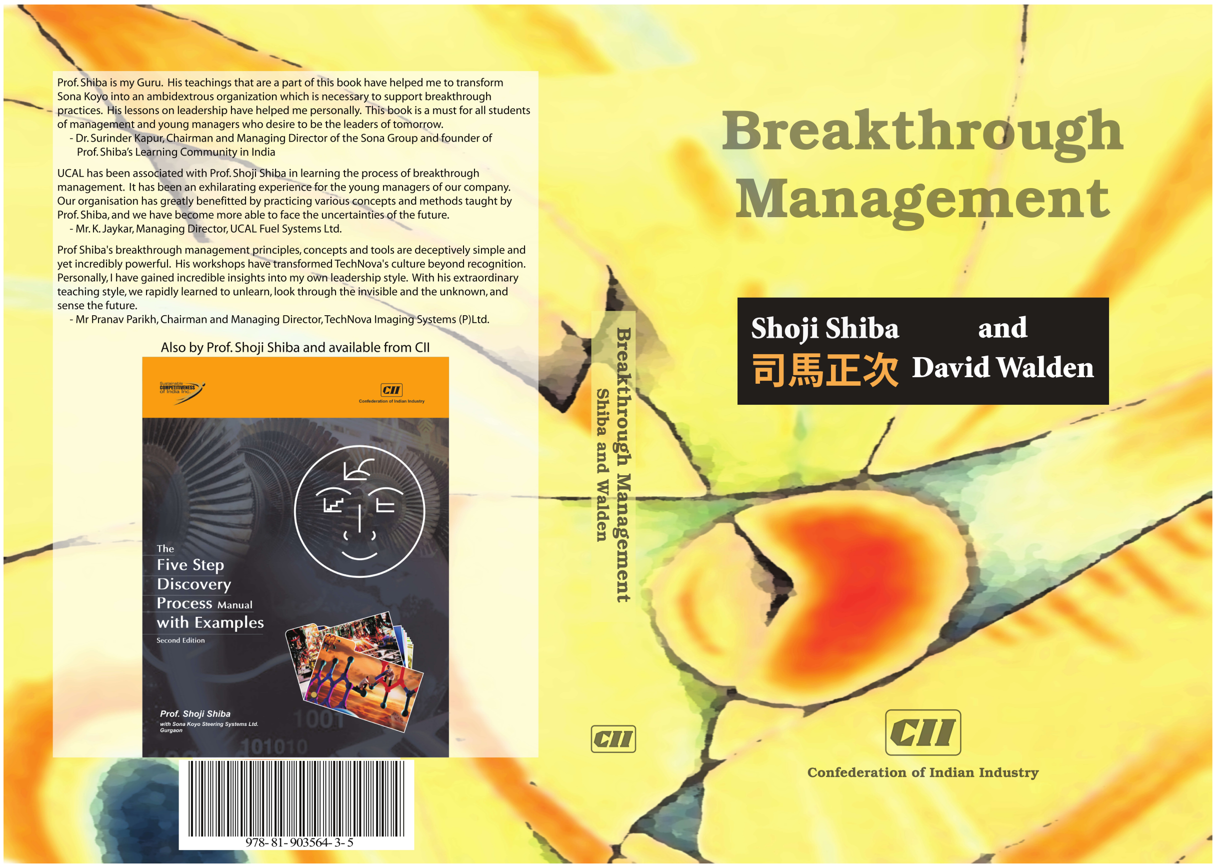
- ☐ publishing and selling via the on-line book stores a compendium of chapters on various aspects of the technical history of the high tech company for which I worked for many years; I am co-editing this book and wrote several chapters
- ☐ republishing and selling via my website (to relatives primarily) an oral history of my mother that my wife edited and which we privately printed a number of years ago using Xerox-copy printing and binding from a thesis binding company
- ☐ ditto for an oral history of my mother-in-law
- ☐ publishing and selling via the on-line book stores a picture book about the salt marsh on which I live and which I have converted to a book from a narrated slide show my son and I developed a number of years ago

A couple of other self-publishing book projects are also in the works.

Disintermediation and the "publisher"

In this note I have been talking about the distinction between a regular publisher and a self-publisher. Technically, as I understand things, the publisher is the place that goes with the ISBN number.

Historically, the place that provided the ISBN number also was the publisher in a much larger sense: the traditional publisher took a hand written or type written manuscript, edited it, designed the book, typeset it, proofread it, perhaps indexed it, had it printed and bound, marketed it, sold it to wholesalers and perhaps retailers, and also perhaps sold foreign rights. These days much of that is outsourced at many publishers and typically the author is expected to do much (or perhaps all) of the marketing. The publisher still does



Prof. Shiba is my Guru. His teachings that are a part of this book have helped me to transform Sona Koyo into an ambidextrous organization which is necessary to support breakthrough practices. His lessons on leadership have helped me personally. This book is a must for all students of management and young managers who desire to be the leaders of tomorrow.
- Dr. Surinder Kapur, Chairman and Managing Director of the Sona Group and founder of Prof. Shiba's Learning Community in India

UCAL has been associated with Prof. Shoji Shiba in learning the process of breakthrough management. It has been an exhilarating experience for the young managers of our company. Our organisation has greatly benefitted by practicing various concepts and methods taught by Prof. Shiba, and we have become more able to face the uncertainties of the future.
- Mr. K. Jaykar, Managing Director, UCAL Fuel Systems Ltd.

Prof Shiba's breakthrough management principles, concepts and tools are deceptively simple and yet incredibly powerful. His workshops have transformed TechNova's culture beyond recognition. Personally, I have gained incredible insights into my own leadership style. With his extraordinary teaching style, we rapidly learned to unlearn, look through the invisible and the unknown, and sense the future.
- Mr Pranav Parikh, Chairman and Managing Director, TechNova Imaging Systems (P)Ltd.

Also by Prof. Shoji Shiba and available from CII

The Five Step Discovery Process Manual with Examples
Second Edition
Prof. Shoji Shiba

978-81-903564-3-5

Breakthrough Management
Shiba and Walden

Breakthrough Management

Shoji Shiba 司馬正次 and David Walden

Confederation of Indian Industry

incur the financial risk of paying for all that work, except in some cases where it expects the author to guarantee enough sales to cover its risk.

The subsidy publisher also traditionally has done all that work except in many cases they don't do any marketing or sales except to the author and his or her friends and family. The subsidy publishers also traditionally have not taken any financial risk, but rather have made a profit by making the author pay for editing, typesetting, printing, etc., perhaps implicitly in a fixed fee per book.

As noted earlier, there have also always been a few self-publishers — people who felt they had something important to say and pay in money or effort to create a pile of books that could be sold or given away.

It seems clear to me that the boundary between publishing and self publishing is becoming more vague. Sure, the entity that provides the ISBN number is technically the publisher, but with all the outsourcing this may not mean much more than providing the ISBN number for some traditional publishers. The whole processing of publishing a book is no longer tightly tied to the entity which provides the ISBN number. The individual who does all the earlier steps him- or herself, or hires them done, is not very different from the traditional publisher except that the traditional publisher has more books in its catalog, perhaps better financing, a reputation as being a legitimate publisher, and thus connections to the world of reviewing.[29]

In the case of *Breakthrough Management*, I did everything myself or paid for it, but I arranged for a non-profit organization in India to provide the ISBN number (it was mutually useful — for them to "publish" my book and for me to have them associated with the book). Am I any less the publisher in fact because the Indian organization is the publisher in name? I don't think so. I did everything including providing ready-to-print PDF files to the printer in India, and I have reprinted the book in other countries without any additional involvement of the Indian organization (but still using its ISBN number, with its permission).

I can imagine that I could take my book to a traditional publisher and, if they like it, make a deal with them whereby they have my ready-to-print files printed and add the book to their catalog in return for appropriate payment of some kind. This happens regularly with foreign rights. For instance, I know that a 2007 Donna Leon mystery book was published in the United States by the US publisher simply buying the ready-to-print files from the original UK publisher and changing the title page and ISBN number.

There is a lot of discussion in the self-publishing world about discrimination against self-publishers. I think this is wasted angst. If it makes sense for non-economic or economic reasons to self-publish, do it. If it makes sense to seek a traditional publisher, do that. Or do a mix: publish books traditional publishers want to publish with them and publish yourself other books you want to publish. Who knows, a good enough self-published book may be picked up by a traditional publisher, and you may be able to just sell the publisher their ready-to-print files. I suspect that at some time in the future traditional publishers will regularly acquire ready-to-print books from their authors as authors seek more control over their work and better financial return, and have more options for skipping traditional publishers altogether. When Stephen King or Tom Clancy decides to hire his own editors, designers, typesetters, etc., and offer their books as ready-to-print files to publishers, I'll bet the publishers will not be able to resist.

Acknowledgments

Some of these notes were originally published in my "Travels in TEX Land column in issue 2007-1 of The PracTEX Journal.[30]. The following people helped me with to those sections: Journal guest editor Yuri Robbers made many helpful suggestions. John Culleton[22] reviewed the content on self-publishing for major errors. Marion Gropen gave me many especially useful suggestions for subtle improvement of various points. Karl Berry reviewed one section for content and spotted many typos throughout the paper. The anonymous reviewers provided helpful corrections, as did TPJ editor Lance Carnes. For the current version of these notes, Frans Goddijn pointed out a number of typos, as did Wybo Dekker.

Biographical sketch

David Walden is retired after a career as an engineer,[31] engineering manager, and general manager involved with research and development of computer and other high tech systems. These days he spends most of his time writing (and now self-publishing). More history is at www.walden-family.com/dave.

Notes

1. *Four Practical Revolutions in Management*, Shoji Shiba and David Walden, Productivity Press, New York, NY, 2003 — www.walden-family.com/4prim.
2. Dan O. Snow, coauthor with Dan Poynter of *U-PUBLISH.COM 4.0: A 'Living Book' to Help You Compete with the Giants of Publishing* has a different point of view. In an email of February 24, 2007, Dan said, "[T]here are nearly *ten times* more outlets for books than bookstores…and they are easier to target, usually pay more, pay faster, and return fewer (if any) unsold books."
3. I know personally of a book where the publisher, John Wiley & Sons, required the authors to purchase this sort of minimum number of copies.
4. See the first paragraph of the section "Printing options" on page 56 for the distinction between offset and digital printing.
5. I suspect this is happening with my book *Four Practical Revolutions in Management*.
6. The royalty percentage figures I use in this paragraph are by way of example. Marion Gropen, who is an expert on the publishing business, in a January 18, 2007, email said:

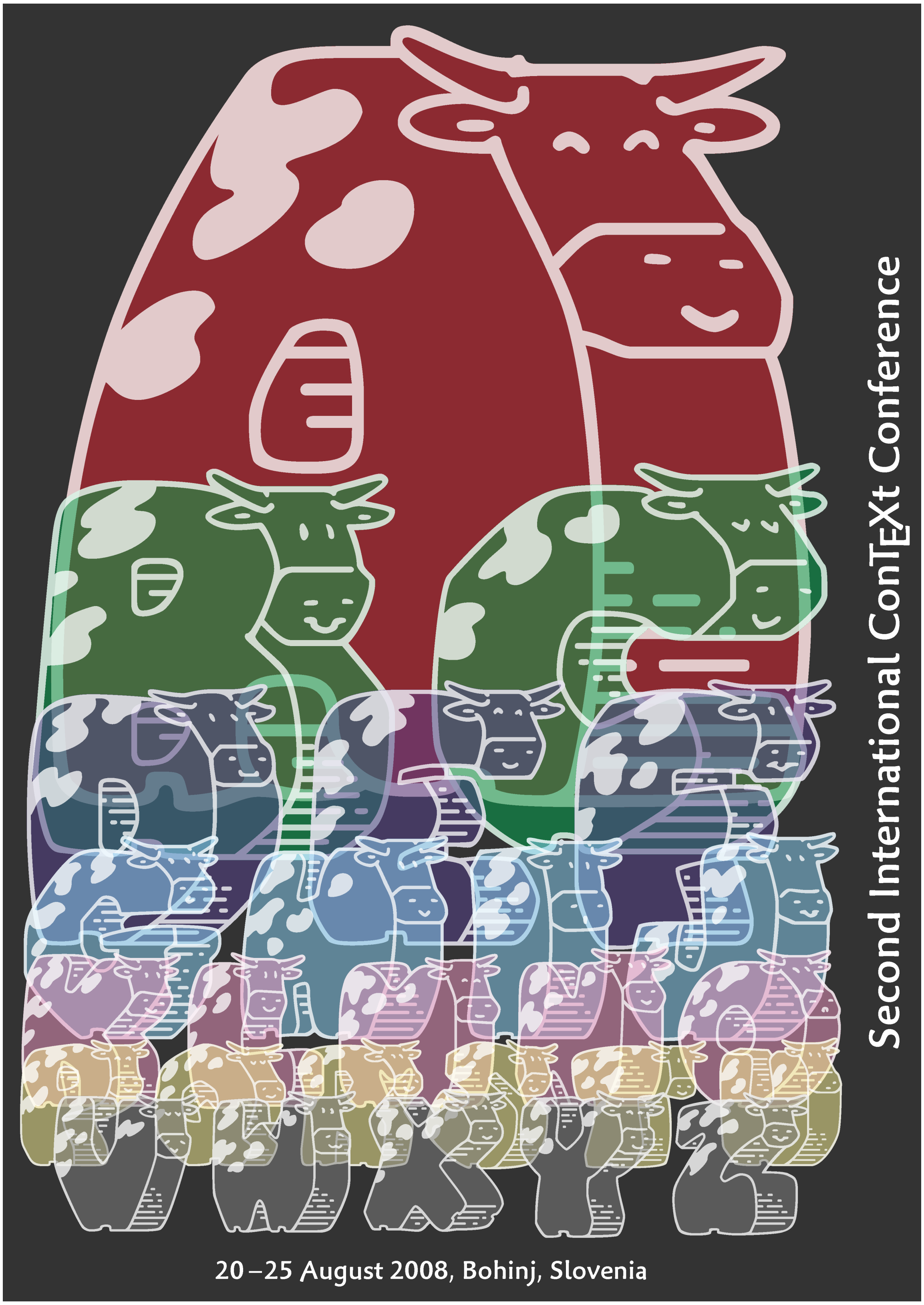
> Royalty rates vary substantially with the type of book. Small houses may pay different rates than the norm; but for larger houses, the norms are practically carved in stone. For example, a trade non-fiction hardback author gets 10 percent of list price (also known as the suggested retail price) for the first 5,000 copies sold, net of returns. The next 5,000 copies yield 12.5 percent of list. And thereafter, the author gets 15 percent of list. Advances are usually calculated to cover something like the expected earnings for the first 6 months of the title's life, although this varies widely.
>
> Mass market fiction tends to run from 5 percent of list to 8 percent. Trade paperbacks are usually between 7 percent and 10 percent. The break points at which the rates step up vary with formats and market segments.
>
> Scientific, medical, technical, professional, and academic publishers generally pay upon net sales (gross sales after discounts and returns). Again, rates and breakpoints will vary depending upon format and market segment.

7. I am sure that there have been some books published using a subsidy publisher which in fact sold lots of books, if the book turned out to be sufficiently interesting to some class of readers. For instance, I have always had the impression that the early editions (ca. 1960) of Marshall Miles' classic and widely read book on contract bridge, *How To Win At Duplicate Bridge*, were published by a subsidy publisher.
8. www.bookmarket.com/selfpublish.html has a nice long list.
9. It helps my motivation to keep writing to have successive drafts of the book have the superficial appearance to being pages from a finished book.
10. I use LaTEX as my typesetting system. Most of what I say in the rest of this note would be just as true for a book typeset with a graphically oriented typesetting system such as InDesign, QuarkExpress, or the free Scribus.

I use LaTEX because it is arguably the most powerful typesetting system available, because it is available without cost,

because it does not use an undocumented proprietary data format that can be obsoleted by newer versions, and because it does not have a corporate goal of selling a new version every year or two that obsoletes prior versions.
11. I created my cover art using Adobe Illustrator (which does get obsoleted periodically with the goal of selling new versions).
12. In retrospect, I believe the printer's quote would have been the same for 20 copies.
13. You will have to Google for the ISBN agency for other countries.
14. I don't have a Massachusettes resale number which is required to avoid paying state sales tax.
15. This was before the dollar got much less valuable compared to the Euro and UK pound.
16. Also, on May 8, 2008, in Houston, Texas, at a meeting of the World Alliance for Quality (an organization of several dozen quality organizations), there was a call for breakthrough projects that would help advance the state of quality worldwide. Sarita Nagpal (deputy directory of CII which provided the ISBN number for our *Breakthrough Management* book), my co-author Shoji Shiba, and I responded to the call by offering to provide the PDF print files for the book free of charge to member organizations so they can print it locally and sell it at prices conducive to widespread circulation in their geographic region. We will see how many organizations take us up on this offer of what I am calling "cooperative printing." Obviously, most self-publishers will choose not to give away potential book sales, but self-publishing allows the freedom to take such an unconventional step.
17. lulu.com is not exactly self-publishing, but this is an inexpensive easy way to get some books in print. For instance, this is the approach TUG president Karl Berry has used to make his *Eplain* and *Fontname* documents available in bound hardcopy format. Lulu actually may have its printing done by another company; I am not sure.
18. Purportedly, LSI's print quality is improving over time as they adopt improved technology.
19. However, my understanding is that self-published books printed by LSI typically do not get in a hard copy Ingram catalog that book stores look at as part of deciding what to order.
20. But see NB on page 55.
21. In addition to following up with the resources listed here, simply google on "self-publishing" and "print on demand" to find pointers to a lot of additional resources.
22. tug.org/interviews/interview-files/john-culleton.html
23. Just after I wrote these words, I started reading the book *Girls Like Us: Carole King, Joni Mitchell, Carly Simon — and the Journal of a Generation* by Sheila Weller (Atria Division of Simon & Shuster, Inc., 2008). On page 8 the right justification looks obviously uneven because there is a lot of end-of-line punctuation. Apparently the typesetter did not use micro-typesetting (which my PDFLaTeX system easily provides) which pushes punctuation slightly into the right margin so the right justification *looks* more even.
24. In addition, the typesetting package I use, LaTeX, does an excellent job (better than many books typeset by "real" publishers) without me having to have so much skill at typesetting. The skill with LaTeX tends to be in the area of modifying the standard templates to get different interior book designs.
25. I also see cover art creation as quite separable (in terms of who does the work) from the design of the interior of a book, although again it can all be done by the same person.
26. I'm not sure how practical it is to draft a book in one of the graphically-oriented typesetting systems. Maybe one has to first draft the book in a text editor or word processor such as Word. I deliberately avoided using Word for drafting. In fact, an important reason for learning about LaTeX was to avoid the clutches of Word. You can read about my experience with this book at tug.org/TUGboat/Articles/tb24-2/tb77walden.pdf.
27. Robert Bringhurst, *The Elements of Typographic Style*, version 3.1, Hartley & Marks Publishers, Vancouver, BC, 2005; James Felici, *The Complete Manual of Typography*, Peachpit Press, Berkeley, CA, 2003.
28. www.lulu.com/titlescorer is fun to try.
29. Somewhere on Dan Poynter's website he says something along of lines that, "You are the publisher of a book if you pay for or do the work to produce the book; if you do several books, you are a small publisher of books; if you do many books, you are a large publisher. In all cases you are a publisher."
30. tug.org/pracjourn/2007-1/walden
31. He was a member of the small team that in 1969 developed ARPANET communications system which evolved into the Internet.

David Walden


Second International ConTEXt Conference
20 – 25 August 2008, Bohinj, Slovenia

MetaPost library project

Introduction
The purpose of this paper is to document the targets and implementation milestones of the MetaPost library project (MPlib). It is intended to serve both as a guideline to the developers and as a monitoring tool for the funding providers.

When it was circulated in the spring of 2007, the MetaPost library proposal paper [1] mentioned two main problem areas that are to be directly tackled by the MPlib project:

- current MetaPost cannot be used as a software component
- the handling of external text labels is becoming outdated

The proposal paper identifies those problem areas and presents a rough split in sub-tasks. The main goal of that paper was to gain funding, and because of that, but also because an in-depth analysis was still missing at that moment in time, it was very short on details.

Immediately after funding was secured, a period of analysis started. That initial information-gathering stage is now finished, and the gained knowledge can be used to further detail the objectives and stages of the project.

Main objectives
The term 'software component' as used above is a loosely defined term. In practice it means that the current MetaPost code should be split into a core library and a client application, with the latter being as small as possible and the former allowing multiple concurrent usage.

Besides the general goal of creating a reusable component on one particular platform, the project has to make sure that MetaPost maintains at least the current level of portability, and that it keeps on having 100% identical behaviour and output across all computer platforms it runs on.

Regardless of implementation details, it is clear that for simplified maintenance and portability, the current code base has to be unified using a single programming language. A literate programming language is desired: we (the TEX community) should do its best to keep the Knuthian tradition of programs that document their implementation alive, whenever that is possible without major extra effort.

Converting the MetaPost core into a library implies adding an indirection layer to the already existing input and output subsystems: component libraries should not interact with the user directly unless explicitly asked to do so. The new to-be-written internal interface will serve to allow the configuration of logging and error handling.

A well-defined and documented interface to the internals of the program is needed to make it possible to add new back-ends as alternatives to the already existing Adobe Postscript generator.

Implementation language
After some consideration, we believe the best choice for the implementation of the core library itself is the C programming language. There are a number of possible arguments to be made in favour of various other languages, but it seems that C has the best overall characteristics for the task at hand:

- there already is a literate programming system available for C that is very close to the current Pascal–based implementation: cweb [2].
- the C language is structurally very similar to pascal, so that identical algorithms can be used in the conversion of the existing code base.
- both the run-time speed and compactness of compiled C code are very high.
- C source code is itself very portable, and C libraries binds easily to other applications and programming languages.
- the market penetration of C is very high, aiding deployment.
- some parts of the current MetaPost are already written in C (although in a non-literate fashion).

Two specific applications of the MPlib core library are planned as part of the current project:

- a command-line C program

  This program will replace the current `mpost` executable, for redistribution in TEX distributions.
- a Lua language binding.

  This will allow the library to be easily integrated into LuaTEX [3], and it will also serve as an example of binding MPlib.

Tasks

From the previous paragraphs, a list of sub-tasks can be derived.

1. **Conversion** and unification of the current sources into cweb.
2. Integrating the hundreds of global variables that are used by the current source into an **single data structure** (instance).
3. The addition of a **indirection layer** for all the input and output streams.
4. Adding an interface for configuration of **error handling**.
5. Writing a **Lua module** to interface to the MetaPost library.
6. Consolidating and documenting an **interface to the internal structures**.
7. Designing and implementing a new high-quality **labelling system**.

Conversion

There are two sub-tasks: the conversion of the current Pascal Web code, and the conversion plus documentation of the current C code, resulting in a unified source.

Conversion of current Pascal code. Because all of the Pascal code uses a pretty small subset of the many possible variations on the basic Pascal language, and because the cweb system itself is nearly identical to Pascal Web, using a machine-assisted approach to the translation is not only feasible but also the most effective.

A Lua script will take care of the conversion from Pascal to C while keeping the Web extensions intact. The choice of the scripting language Lua [4] is based on three considerations:

1. Efficiency of the conversion program is not relevant, as it will be used only once.

   Therefore, there is no reason to choose for a compiled language like C, and scripting languages allow for a much faster development cycle.
2. Lua has a module that allows script programs to be written that use Parsing Expression Grammars [5, 6] to parse input.

   This makes it very easy to write a proper parser for web-based programs like MetaPost.
3. Familiarity with the language.

   It would not be worth learning a different programming language just to do this conversion step, and of the languages Taco knows, Lua is best suited to the task at hand.

Conversion of current C code. The main MetaPost program is written in Pascal Web, but some (about 5000 lines, mostly the font inclusion) is written directly in C code. This code is not currently not written in a literate programming fashion, and it will be worthwhile to do that conversion. This has to be done by hand, but the amount of code is not that large and it will help to create a more consistent distribution and build process.

Single data structure

For a code library to be as widely usable as possible, it should not use any non-local variables. Instead, a code library normally defines a single function, that will allocate and return a data structure that will from then on be passed on from one library function to the next until the application is done with it.

The current internals have to reworked extensively to allow this: MetaPost as it stands uses about three hundred global variables. Some of these are just for internal communication between functions (a side–effect of limitations in Pascal Web), and some are just global constant values, but most are used to store the internal state of the MetaPost engine.

Actions needed:

- all of the globals have to be incorporated into a big single structure.
- nearly all functions have to be altered to use this structure are the first argument.
- the memory `dump/undump` code has to be rewritten to handle this new data model correctly.

When this task is complete, the cweb code can provisionally be split into a library and an application.

For the moment, the Knuthian heap-based memory management and string pool handling will be converted to C without functional changes, but at a later moment these bits may be converted to use dynamic memory allocation throughout.

Indirection layer

We envisage that it will be possible to register callback functions that will make it possible to override default behaviour that is very close or identical to the current state.

The advantage of setting it up in this way is that application programs that want to be compatible with the way the old `mpost` executable can be quite small, and for the moment at least, this seems desirable.

Error handling

To allow the user to configure the error handling, first the existing error handling code has to be unified so that all the possible errors pass through the same interface. After that is done, it will be possible to add a callback there to allow configurability.

Lua module
At the base functionality level, this is a straightforward process of interfacing C functions to Lua scripting, that can handled almost totally in an automated fashion by already existing tools like [7].

But optimization of the user interface so that it is easy to use has to be done manually. Experiences has shown that C libraries almost always are too low-level for a scripting language, and the necessary glue code has to be written by hand.

Because much of this glue code will itself depend on user feedback, it follows that while a first version of Lua module will probably be available at the time of the first release of MPlib(summer 2008), the module will probably not be finalized until quite time afterward.

Interface to the internal structures
While we are strictly dealing with the problem of how to use MetaPost as a component, it is sufficient to just have a 'constructor' function, a 'destructor' function, and a 'run' function.

But that is not good enough if we want to alter the program, for example by adding a new back-end. For this, it is necessary to have access to at least some of the actual internals of the program.

For such interfaces to be useful, a bit of internal cleanup is needed:

- Some of the interface structures are not as clean cut as they could be (or should be, according to current practice.

  Some cleanup is needed to provide a clean front.
- most of the required documentation is already present in the Pascal Web source, but not presented in the best possible way.

  This documentation should be copied out of the source and placed in a separate manual.

Labelling system
When all of the above are complete, it is time to think about a new external labelling system to replace `btex ... etex`.

New insights are expected to come from the use of MPlib inside LuaTeX, so writing this section now would be premature.

Time line
The development of MPlib is expected to progress linearly through the first few point, up to first basic 'Lua module' implementation. After that, work will continue more or less simultaneous on the remaining items.

The stage 'single data structure' is expected to be complete around end of February, and everything up to the basic 'Lua module' will be presented at the Cork TUG meeting in the summer of 2008.

References

[1] Hans Hagen and Taco Hoekwater. *Metapost library proposal*. The Netherlands, 2007.
[2] Donald E. Knuth and Silvio Levy. *The CWEB System of Structured Documentation, Version 3.0*. Addison-Wesley, Reading, MA, USA, 1993.
[3] Hans Hagen and Taco Hoekwater.
http://www.luatex.org
The official LuaTeX website.
[4] Roberto Ierusalimschy, Waldemar Celes and Luiz Henrique de Figueiredo.
http://www.lua.org
The official Lua website.
[5] Bryan Ford.
http://pdos.csail.mit.edu/~baford/packrat/
This website explains the theory behind Parsing Expression Grammars, and Packrat Parsing in general.
[6] Roberto Ierusalimschy.
http://www.inf.puc-rio.br/~roberto/lpeg.html
The homepage of the Lua module that implements PEGs.
[7] Waldemar Celes.
http://www.techgraf.puc-rio.br/~celes/tolua
The homepage of toLua, a program that generates C/C++ to Lua bindings.

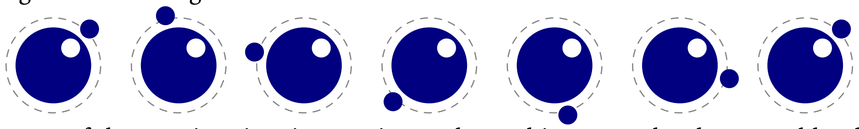
Taco Hoekwater and Hans Hagen

The MetaPost Library

Introduction

If MetaPost support had not been as tightly integrated into ConTEXt as it is, at least half of the projects Pragma ADE has been doing in the last decade could not have been done at all. Take for instance backgrounds behind text or graphic markers alongside text. These are probably the most complex mechanisms in ConTEXt: positions are stored, and positional information is passed on to MetaPost, where intersections between the text areas and the running text are converted into graphics that are then positioned in the background of the text. Underlining of text (sometimes used in the educational documents that we typeset) and change bars (in the margins) are implemented using the same mechanism because those are basically a background with only one of the frame sides drawn.

You can probably imagine that a 300 page document with several such graphics per page takes a while to process. A nice example of such integrated graphics is the LuaTEX reference manual, that has an unique graphic at each page: a stylized image of a revolving moon.



Most of the running time integrating such graphics seemed to be caused by the mechanics of the process: starting the separate MetaPost interpreter and having to deal with a number of temporary files. Therefore our expectations were high with regards to integrating MetaPost more tightly into LuaTEX. Besides the speed gain, it also true that the simpler the process of using such use of graphics becomes, the more modern a TEX runs looks and the less problems new users will have with understanding how all the processes cooperate.

This article will not discuss the application interface of the MPlib library in detail, for that there is the LuaTEX manual. In short, using the embedded MetaPost interpreter in LuaTEX boils down to the following:

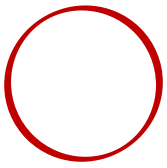
- Open an instance using `mplib.new`, either to process images with a format to be loaded, or to create such a format. This function returns a library object.
- Execute sequences of MetaPost commands, using the object's `execute` method. This returns a result.
- Check if the result is valid and (if it is okay) request the list of objects. Do whatever you want with them, most probably convert them to some output format. You can also request a string representation of a graphic in PostScript format.

There is no need to close the library object. As long as you didn't make any fatal errors, the library recovers well and can stay alive during the entire LuaTEX run.

Support for MPlib depends on a few components: integration, conversion and extensions. This article shows some of the code involved in supporting the library. Let's start with the conversion.

Conversion

The result of a MetaPost run traditionally is a PostScript language description of the generated graphic(s). When pdf is needed, that PostScript code has to be converted to the target format. This includes embedded text as well as penshapes used for drawing. To demonstrate, here is a simple example graphic:



```
draw fullcircle
  scaled 2cm
  withpen pencircle xscaled 1mm yscaled .5mm rotated 30
  withcolor .75red ;
```

Notice how the pen is not a circle but a rotated ellipse. Later on it will become clear what the consequences of that are for the conversion.

How does this output look in PostScript? If the preamble is left out it looks like this:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: -30 -30 30 30
%%HiResBoundingBox: -29.624 -29.28394 29.624 29.28394
%%Creator: MetaPost 1.003
%%CreationDate: 2008.05.15:1534
%%Pages: 1
% <<stripped preamble was here>>
%%Page: 1 1
 0.75 0 0 R 2.55513 hlw rd 1 lj 10 ml
q n 28.34645 0 m
28.34645 7.51828 25.35938 14.72774 20.04356 20.04356 c
14.72774 25.35938 7.51828 28.34645 0 28.34645 c
-7.51828 28.34645 -14.72774 25.35938 -20.04356 20.04356 c
-25.35938 14.72774 -28.34645 7.51828 -28.34645 0 c
-28.34645 -7.51828 -25.35938 -14.72774 -20.04356 -20.04356 c
-14.72774 -25.35938 -7.51828 -28.34645 0 -28.34645 c
7.51828 -28.34645 14.72774 -25.35938 20.04356 -20.04356 c
25.35938 -14.72774 28.34645 -7.51828 28.34645 0 c p
 [0.96077 0.5547 -0.27734 0.4804 0 0] t S Q
P
%%EOF
```

The most prominent code here concerns the path. The numbers in brackets define the transformation matrix for the pen we used. The pdf variant looks as follows:

```
q
0.750 0.000 0.000 rg 0.750 0.000 0.000 RG
10.000000 M
1 j
1 J
2.555120 w
q
0.960769 0.554701 -0.277351 0.480387 0.000000 0.000000 cm
22.127960 -25.551051 m
25.516390 -13.813203 26.433849 0.135002 24.679994 13.225878 c
22.926120 26.316745 18.644486 37.478783 12.775526 44.255644 c
6.906565 51.032505 -0.067572 52.867453 -6.613036 49.359793 c
```

```
-13.158495 45.852096 -18.739529 37.288899 -22.127960 25.551051 c
-25.516390 13.813203 -26.433849 -0.135002 -24.679994 -13.225878 c
-22.926120 -26.316745 -18.644486 -37.478783 -12.775526 -44.255644 c
-6.906565 -51.032505 0.067572 -52.867453 6.613036 -49.359793 c
13.158495 -45.852096 18.739529 -37.288899 22.127960 -25.551051 c
h S
Q
0 g 0 G
Q
```

The operators don't look much different from the PostScript, which is mostly due to the fact that in the PostScript code, the preamble defines shortcuts like `c` for `curveto`. Again, most code involves the path. However, this time the numbers are different and the transformation comes before the path.

In the case of pdf output, we could use TEX itself to do the conversion: a generic converter is implemented in `supp-pdf.tex`, while a converter optimized for ConTEXt MkII is defined in the files whose names start with `meta-pdf`. But in ConTEXt MkIV we use Lua code for the conversion instead. Thanks to Lua's powerful lpeg parsing library, this gives cleaner code and is also faster. This converter currently lives in `mlib-pdf.lua`.

Now, with the embedded MetaPost library, conversion goes different still because now it is possible to request the drawn result and associated information in the form of Lua tables.

```
figure={
 ["boundingbox"]={
  ["llx"]=-29.623992919922,
  ["lly"]=-29.283935546875,
  ["urx"]=29.623992919922,
  ["ury"]=29.283935546875,
 },
 ["objects"]={
  {
   ["color"]={ 0.75, 0, 0 },
   ["linecap"]=1,
   ["linejoin"]=1,
   ["miterlimit"]=10,
   ["path"]={
    {
     ["left_x"]=28.346450805664,
     ["left_y"]=-7.5182800292969,
     ["right_x"]=28.346450805664,
     ["right_y"]=7.5182800292969,
     ["x_coord"]=28.346450805664,
     ["y_coord"]=0,
    },
    {
     ["left_x"]=25.359375,
     ["left_y"]=14.727737426758,
     ["right_x"]=14.727737426758,
     ["right_y"]=25.359375,
     ["x_coord"]=20.043563842773,
     ["y_coord"]=20.043563842773,
    },
    {
     ["left_x"]=7.5182800292969,
     ["left_y"]=28.346450805664,
     ["right_x"]=-7.5182800292969,
     ["right_y"]=28.346450805664,
     ["x_coord"]=0,
     ["y_coord"]=28.346450805664,
    },
    {
     ["left_x"]=-14.727737426758,
     ["left_y"]=25.359375,
     ["right_x"]=-25.359375,
     ["right_y"]=14.727737426758,
     ["x_coord"]=-20.043563842773,
     ["y_coord"]=20.043563842773,
    },
    {
     ["left_x"]=-28.346450805664,
     ["left_y"]=7.5182800292969,
     ["right_x"]=-28.346450805664,
     ["right_y"]=-7.5182800292969,
     ["x_coord"]=-28.346450805664,
     ["y_coord"]=0,
    },
    {
     ["left_x"]=-25.359375,
     ["left_y"]=-14.727737426758,
     ["right_x"]=-14.727737426758,
     ["right_y"]=-25.359375,
     ["x_coord"]=-20.043563842773,
     ["y_coord"]=-20.043563842773,
    },
    {
     ["left_x"]=-7.5182800292969,
     ["left_y"]=-28.346450805664,
     ["right_x"]=7.5182800292969,
     ["right_y"]=-28.346450805664,
     ["x_coord"]=0,
     ["y_coord"]=-28.346450805664,
    },
    {
     ["left_x"]=14.727737426758,
     ["left_y"]=-25.359375,
     ["right_x"]=25.359375,
     ["right_y"]=-14.727737426758,
     ["x_coord"]=20.043563842773,
     ["y_coord"]=-20.043563842773,
    },
   },
   ["pen"]={
    {
     ["left_x"]=2.4548797607422,
     ["left_y"]=1.4173278808594,
     ["right_x"]=-0.70866394042969,
     ["right_y"]=1.2274475097656,
     ["x_coord"]=0,
     ["y_coord"]=0,
    },
    ["type"]="elliptical",
   },
   ["type"]="outline",
  },
 },
}
```

This means that instead of parsing PostScript output, we now can operate on a proper datastructure and get code like the following:

```
function convertgraphic(result)
  if result then
    local figures = result.fig
    if figures then
      for fig in ipairs(figures) do
        local llx, lly, urx, ury = unpack(fig:boundingbox())
        if urx > llx then
          startgraphic(llx, lly, urx, ury)
          for object in ipairs(fig:objects()) do
            if object.type == "..." then
              ...
              flushgraphic(...)
              ...
            else
              ...
            end
          end
          finishgraphic()
        end
      end
    end
  end
end
```

Here `result` is what the library returns when one or more graphics are processed. As you can deduce from this snippet, a result can contain multiple figures. Each figure corresponds with a `beginfig ... endfig`. The graphic operators that the converter generates (so called pdf literals) have to be encapsulated in a proper box so this is why we have:

- `startgraphic`: start packaging the graphic
- `flushgraphic`: pipe literals to TEX
- `finishgraphic`: finish packaging the graphic

It does not matter what number you passed to `beginfig`, the graphics come out in the natural order.

Little over half a dozen different object types are possible. The example MetaPost `draw` command from above results in an `outline` object. This object contains not only path information but also carries rendering data, like the color and the pen. So, in the end we will flush code like `1 M` which sets the `miterlimit` to one or `.5 g` which sets the color to 50% gray, in addition to a path.

Because objects are returned in a way that closely resembles a MetaPost's internals, some extra work needs to be done in order to calculate paths with elliptical pens. An example of a helper function in somewhat simplified form is shown next:

```
function pen_characteristics(object)
  local p = object.pen[1]
  local wx, wy, width
  if p.right_x == p.x_coord and p.left_y == p.y_coord then
    wx = abs(p.left_x  - p.x_coord)
    wy = abs(p.right_y - p.y_coord)
  else -- pyth: sqrt(a^2 +b^2)
```

```
    wx = pyth(p.left_x - p.x_coord, p.right_x - p.x_coord)
    wy = pyth(p.left_y - p.y_coord, p.right_y - p.y_coord)
  end
  if wy/coord_range_x(object.path, wx) >=
                  wx/coord_range_y(object.path, wy) then
    width = wy
  else
    width = wx
  end
  local sx, sy = p.left_x, p.right_y
  local rx, ry = p.left_y, p.right_x
  local tx, ty = p.x_coord, p.y_coord
  if width ~= 1 then
    if width == 0 then
      sx, sy = 1, 1
    else
      rx, ry, sx, sy = rx/width, ry/width, sx/width, sy/width
    end
  end
  if abs(sx) < eps then sx = eps end
  if abs(sy) < eps then sy = eps end
  return sx, rx, ry, sy, tx, ty, width
end
```

If sx and sy are 1, there is no need to transform the path, otherwise a suitable transformation matrix is calculated and returned. The function itself uses a few helpers that make the calculations even more obscure. This kind of code does not fall in the category trivial and as already mentioned, these basic algorithms were derived from the MetaPost sources. Even so, these snippets demonstrate that interfacing using Lua does not look that bad.

In the actual MkIV code things look a bit different because it does a bit more and uses optimized code. There you will also find the code dealing with the actual transformation, of which these helpers are just a portion.

If you compare the PostScript and the pdf code you will notice that the paths looks different. This is because the use and application of a transformation matrix in pdf is different from how it is handled in PostScript. In pdf more work is assumed to be done by the pdf generating application. This is why in both the TEX and the Lua based converters you will find transformation code and the library follows the same pattern. In that respect pdf differs fundamentally from PostScript.

Within the TEX based converter there was the problem of keeping the needed calculations within TEX's accuracy, which fortunately permits larger values that MetaPost can produce. This plus the parsing code resulted in a not-that-easy to follow bunch of TEX code. The Lua based parser is more readable, but since it also operates on PostScript code it is kind of unnatural too, but at least there are less problems with keeping the calculations sane. The MPlib based converter is definitely the cleanest and least sensitive to future changes in the PostScript output. Does this mean that there is no ugly code left? Alas, as we will see in the next section, dealing with extensions is still somewhat messy. In practice users will not be bothered with such issues, because writing a converter is a one time job by macro package writers.

Extensions

In MetaFun, which is the MetaPost format used with ConTEXt, a few extensions are provided, like:

- cmyk, spot and multitone colors
- including external graphics
- lineair and circulair shades
- texts converted to outlines
- inserting arbitrary texts

Until now, most of these extensions have been implemented by using specially coded colors and by injecting so called specials (think of them as comments) into the output. On one of our trips to a TEX conference, we discussed ways to pass information along with paths and eventually we arrived at associating text strings with paths as a simple and efficient solution. As a result, recently MetaPost was extended by `withprescript` and `withpostscript` directives. For those who are unfamiliar with these new scripts, they are used as follows:

```
draw fullcircle withprescript "hello" withpostscript "world" ;
```

In the PostScript output these scripts end up before and after the path, but in the pdf converter they can be overloaded to implement extensions, and that works reasonably well. However, at the moment there cannot be multiple pre- and postscripts associated with a single path inside the MetaPost internals. This means that for the moment, the scripts mechanism is only used for a few of the extensions. Future versions of MPlib may provide more sophisticated methods for carrying information around.

The MkIV conversion mechanism uses scripts for graphic inclusion, shading and text processing but unfortunately cannot use them for more advanced color support.

A nasty complication is that the color spaces in MetaPost don't cast, which means that one cannot assign any color to a color variables: each colorspace has it's own type of variable.

```
color     one ; one := (1,1,0)   ; % correct
cmykcolor two ; two := (1,0,0,1) ; % correct
one := two ; % error
fill fullcircle scaled 1cm withcolor .5[one,two] ; % error
```

In ConTEXt we use constructs like this:

```
\startreusableMPgraphic{test}
  fill fullcircle scaled 1cm withcolor \MPcolor{mycolor} ;
\stopreusableMPgraphic

\reuseMPgraphic{test}
```

Because `withcolor` is clever enough to understand what color type it receives, this is ok, but how about:

```
\startreusableMPgraphic{test}
  color c ; c := \MPcolor{mycolor} ;
  fill fullcircle scaled 1cm withcolor c ;
\stopreusableMPgraphic
```

Here the color variable only accepts an rgb color and because in ConTEXt there is mixed color space support combined with automatic colorspace conversions, it doesn't know in advance what type it is going to get. By implementing color spaces other than rgb using special colors (as before) such type mismatches can be avoided.

The two techniques (coding specials in colors and pre/postscripts) cannot be combined because a script is associated with a path and cannot be bound to a variable like c. So this again is an argument for using special colors that remap onto cmyk spot or multi-tone colors.

Another area of extensions is text. In previous versions of ConTEXt the text processing was already isolated: text ended up in a separate file and was processed in an separate run. More recent versions of ConTEXt use a more abstract model of boxes that are preprocessed before a run, which avoids the external run(s). In the new approach everything can be kept internal. The conversion even permits constructs like:

```
for i=1 upto 100 :
  draw btex oeps etex rotated i ;
endfor ;
```

but since this construct is kind of obsolete (at least in the library version of MetaPost) it is better to use:

```
for i=1 upto 100 :
  draw textext("cycle " & decimal i) rotated i ;
endfor ;
```

Internally a trial pass is done so that indeed 100 different texts will be drawn. The throughput of texts is so high that in practice one will not even notice that this happens.

Dealing with text is yet another example of using lpeg. The following snippet of code sheds some light on how text in graphics is dealt with. Actually this is a variation on a previous implementation. That one was slightly faster but looked more complex. It was also not robust for complex texts defined in macros in a format.

```
local P, S, V, Cs = lpeg.P, lpeg.S, lpeg.V, lpeg.Cs

local btex    = P("btex")
local etex    = P(" etex")
local vtex    = P("verbatimtex")
local ttex    = P("textext")
local gtex    = P("graphictext")
local spacing = S(" \n\r\t\v")^0
local dquote  = P('"')

local found = false

local function convert(str)
  found = true
  return "textext(\"" .. str .. "\")"
end
local function ditto(str)
  return "\" & ditto & \""
end
local function register()
```

```
  found = true
end

local parser = P {
    [1] = Cs((V(2)/register + V(3)/convert + 1)^0),
    [2] = ttex + gtex,
    [3] = (btex + vtex) * spacing *
                Cs((dquote/ditto + (1-etex))^0) * etex,
}

function metapost.check_texts(str)
  found = false
  return parser:match(str), found
end
```

If you are unfamiliar with lpeg it may take a while to see what happens here: we replace the text between `btex` and `etex` by a call to `textext`, a macro. Special care is given to embedded double quotes.

When text is found, the graphic is processed two times. The definition of `textext` is different for each run. The first run we have:

```
vardef textext(expr str) =
    image (
        draw unitsquare
            withprescript "tf"
            withpostscript str ;
    )
enddef ;
```

After the first run the result is not really converted, but just the outlines with the `tf` prescript are filtered. In the loop over the object there is code like:

```
local prescript = object.prescript
if prescript then
  local special = metapost.specials[prescript]
  if special then
    special(object.postscript,object)
  end
end
```

Here, `metapost` is just the namespace used by the converter. The prescript tag `tf` triggers a function:

```
function metapost.specials.tf(specification,object)
  tex.sprint(tex.ctxcatcodes,format("\\MPLIBsettext{%s}{%s}",
    metapost.textext_current,specification))
  if metapost.textext_current < metapost.textext_last then
    metapost.textext_current = metapost.textext_current + 1
  end
  ...
end
```

Again, you can forget about the details of this function. Important is that there is a call out to TEX that will process the text. Each snippet gets the number of the box that holds the content. The macro that is called just puts stuff in a box:

```
\def\MPLIBsettext#1#2%
  {\global\setbox#1\hbox{#2}}
```

In the next processing cycle of the MetaPost code, the `textext` macro does something different :

```
vardef textext(expr str) =
    image (
        _tt_n_ := _tt_n_ + 1 ;
        draw unitsquare
            xscaled _tt_w_[_tt_n_]
            yscaled (_tt_h_[_tt_n_] + _tt_d_[_tt_n_])
            withprescript "ts"
            withpostscript decimal _tt_n_ ;
    )
enddef ;
```

This time the by then known dimensions of the box that is used to store the snippet are used. These are stored in the `_tt_w_`, `_tt_h_` and `_tt_d_` arrays. The arrays are defined by Lua using information about the boxes, and passed to the library before the second run. The result from the second MetaPost run is converted, and again the prescript is used as trigger:

```
function metapost.specials.ts(specification,object,result)
    local op = object.path
    local first, second, fourth  = op[1], op[2], op[4]
    local tx, ty = first.x_coord      , first.y_coord
    local sx, sy = second.x_coord - tx, fourth.y_coord - ty
    local rx, ry = second.y_coord - ty, fourth.x_coord - tx
    if sx == 0 then sx = 0.00001 end
    if sy == 0 then sy = 0.00001 end
    metapost.flushfigure(result)
    tex.sprint(tex.ctxcatcodes,format(
        "\\MPLIBgettext{%f}{%f}{%f}{%f}{%f}{%f}{%s}",
        sx,rx,ry,sy,tx,ty,metapost.textext_current))
    ...
end
```

At this point the converter is actually converting the graphic and passing pdf literals to TEX. As soon as it encounters a text, it flushes the pdf code collected so far and injects some TEX code. The TEX macro looks like:

```
\def\MPLIBgettext#1#2#3#4#5#6#7%
  {\ctxlua{metapost.sxsy(\number\wd#7,\number\ht#7,\number\dp#7)}%
   \pdfliteral{q #1 #2 #3 #4 #5 #6 cm}%
   \vbox to \zeropoint{\vss\hbox to \zeropoint
     {\scale[sx=\sx,sy=\sy]{\raise\dp#7\box#7}\hss}}%
   \pdfliteral{Q}}
```

Because text can be transformed, it needs to be scale back to the right dimensions, using both the original box dimensions and the transformation of the unitquare associated with the text.

```
local factor = 65536*(7200/7227)
```

```
function metapost.sxsy(wd,ht,dp) -- helper for text
  commands.edef("sx",(wd ~= 0 and 1/( wd    /(factor))) or 0)
  commands.edef("sy",(wd ~= 0 and 1/((ht+dp)/(factor))) or 0)
end
```

So, in fact there are the following two processing alternatives:

- tex: calls a Lua function that processed the graphic
- lua: parse the MetaPost code for texts and decide if two runs are needed

Now, if there was no text to be found, the continuation is:

- lua: process the code using the library
- lua: convert the resulting graphic (if needed) and check if texts are used

Otherwise, the next steps are:

- lua: process the code using the library
- lua: parse the resulting graphic for texts (in the postscripts) and signal TEX to process these texts afterwards
- tex: process the collected text and put the result in boxes
- lua: process the code again using the library but this time let the unitsquare be transformed c.c. the text dimensions
- lua: convert the resulting graphic and replace the transformed unitsquare by the boxes with text

The processor itself is used in the MkIV graphic function that takes care of the multiple passes mentioned before. To give you an idea of how it works, here is how the main graphic processing function roughly looks.

```
local current_format, current_graphic

function metapost.graphic_base_pass(mpsformat,str,preamble)
    local prepared, done = metapost.check_texts(str)
    metapost.textext_current = metapost.first_box
    if done then
        current_format, current_graphic = mpsformat, prepared
        metapost.process(mpsformat, {
            preamble or "",
            "beginfig(1); ",
            "_trial_run_ := true ;",
            prepared,
            "endfig ;"
        }, true ) -- true means: trialrun
        tex.sprint(tex.ctxcatcodes,
            "\\ctxlua{metapost.graphic_extra_pass()}")
    else
        metapost.process(mpsformat, {
            preamble or "",
            "beginfig(1); ",
            "_trial_run_ := false ;",
            str,
            "endfig ;"
        } )
    end
```

```
end

function metapost.graphic_extra_pass()
    metapost.textext_current = metapost.first_box
    metapost.process(current_format, {
        "beginfig(0); ",
        "_trial_run_ := false ;",
        table.concat(metapost.text_texts_data()," ;\n"),
        current_graphic,
        "endfig ;"
    })
end
```

The box information is generated as follows:

```
function metapost.text_texts_data()
    local t, n = { }, 0
    for i = metapost.first_box, metapost.last_box do
        n = n + 1
        if tex.box[i] then
            t[#t+1] = format(
                "_tt_w_[%i]:=%f;_tt_h_[%i]:=%f;_tt_d_[%i]:=%f;",
                n,tex.wd[i]/factor,
                n,tex.ht[i]/factor,
                n,tex.dp[i]/factor
            )
        else
            break
        end
    end
    return t
end
```

This is a typical example of accessing information available inside TEX from Lua, in this case information about boxes.

The `trial_run` flag is used at the MetaPost end, in fact the `textext` macro looks as follows:

```
vardef textext(expr str) =
    if _trial_run_ :
        % see first variant above
    else :
        % see second variant above
    fi
enddef ;
```

This trickery is not new. We used it already in ConTEXt for some time, but until now the multiple runs took way more time and from the perspective of the user this all looked much more complex.

It may not be that obvious, but: in the case of a trial run (for instance when texts are found), after the first processing stage, and during the parsing of the result, the commands that typeset the content will be printed to TEX. After processing, the command to do an extra pass is printed to TEX also. So, once control is passed back to TEX, at some point TEX itself will pass control back to Lua and do the extra pass.

The base function is called in:

```
function metapost.graphic(mpsformat,str,preamble)
    local mpx = metapost.format(mpsformat or "metafun")
    metapost.graphic_base_pass(mpx,str,preamble)
end
```

The `metapost.format` function is part of `mlib-run`. It loads the `metafun` format, possibly after (re)generating it.

Now, admittedly all this looks a bit messy, but in pure TEX macros it would be even more so. Sometime in the future, the postponed calls to `\ctxlua` and the explicit `\pdfliterals` can and will be replaced by using direct node generation, but that requires a rewrite of the internal LuaTEX support for pdf literals.

The snippets are part of the `mlib-*` files of MkIV. These files are tagged as experimental and will stay that way for a while yet.

Summarizing the impact of MPlib on extensions, we can conclude that some are done better and some more or less the same. There are some conceptual problems that prohibit using pre- and postscripts for everything (at least currently).

Integrating

The largest impact of MPlib is processing graphics at runtime. In MkII there are two methods: real runtime processing (each graphic triggered a call to MetaPost) and collective processing (between TEX runs). The first method slows down the TEX run, the second method generates a whole lot of intermediate PostScript files. In both cases there is a lot of file io involved.

In MkIV, the integrated library is capable of processing thousands of graphics per second, including conversion. The preliminary tests (which involved no extensions) involved graphics with 10 random circles drawn with penshapes in random colors, and the thoughput was around 2000 such graphics per second on a 2.3 MHz Core Duo:

In practice there will be some more overhead involved than in the tests. For instance, in ConTEXt information about the current state of TEX has to be passed on also: page dimensions, font information, typesetting related parameters, preamble code, etc.

The whole TEX interface is written around one process function:

```
metapost.graphic(metapost.format("metafun"),"mp code")
```

optionally a preamble can be passed as the third argument. This one function is used in several other macros, like:

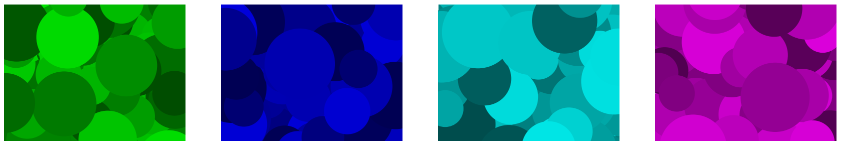
```
\startMPcode                       ... \stopMPcode
\startMPpage                       ... \stopMPpage
\startuseMPgraphic      {name} ... \stopuseMPgraphic
\startreusableMPgraphic{name} ... \stopreusableMPgraphic
\startuniqueMPgraphic  {name} ... \stopuniqueMPgraphic
```

```
\useMPgraphic{name}
\reuseMPgraphic{name}
\uniqueMPgraphic{name}
```

The user interface is downward compatible: in MkIV the same top-level commands are provided as in MkII. However, the (previously required) configuration macros and flags are obsolete.

This time, the conclusion is that the impact on ConTeXt is immense: The code for embedding graphics is very clean, and the running time for graphics inclusion is now negligible. Support for text in graphics is more natural now, and takes no runtime either (in MkII some parsing in TeX takes place, and if needed long lines are split; all this takes time).

In the styles that Pragma ADE uses internally, there is support for the generation of placeholders for missing graphics. These placeholders are MetaPost graphics that have some 60 randomly scaled circles with randomized colors. The time involved in generating 50 such graphics is (on Hans' machine) some 14 seconds, while in LuaTeX only half a second is needed.



Because LuaTeX needs more startup time and deals with larger fonts resources, pdfTeX is generally faster, but now that we have MPlib, LuaTeX suddenly is the winner.

Hans Hagen
Taco Hoekwater

Reshaping Euler
A collaboration with Hermann Zapf

Keywords
fonts, euler, update

It is no secret that over the last few years Hermann Zapf has been reshaping some of his designs, most notably Palatino and Optima. Some three years ago, when Volker and Hans were talking to Hermann, they discovered he would like to improve the Euler fonts as well. These fonts were developed a few decades ago using the technology of those days, in close cooperation between Don Knuth and Hermann Zapf.

The glyphs were drawn on paper about 6cm height and these drawings were digitized using pinpoints on paper with a raster. The resulting points were translated to Metafont and some additional math shapes were added afterwards. Later, when the fonts became popular with TEXies, virtual fonts were created using Euler and AMS Math fonts.

The resulting bitmap fonts were fine for the bitmap-oriented TEX backends of those days. Later, when bitmaps became outdated, the bitmaps became outlines, and the artifacts introduced in the digitization became somewhat more prominent especially when the fonts were scaled).

The reasons why Hermann wanted to reshape Euler were manifold. First, he wanted to improve some details related to drawing with a broad pen. Then, the slope as well as the descenders of some glyphs needed to be adapted. The strokes had to be made more consistent too. Finally, the characters that were not Euler (but had been added afterwards) had to be redrawn: first the core characters, later (in principle) all characters that TEXies use. This last effort is still on the agenda and part of making Euler Unicode compliant.

When we met Hermann on a subsequent occasion, the topic of reshaping Euler came up again, and we decided to go ahead with an active project. Taco was willing to join in and we decided to improve the fonts by just editing the Type 1 fonts.

Because the project would take more than a year (at that time Hermann was still working at Linotype on his other projects), we decided to make this redesign into a present for Don Knuth's 70th birthday. At that point the old Euler was 25 years old.

The following graphics display some of the changes. Some are more prominent than others. Even small corrections help improve the overall look and feel because they influence our perception of black on white. (It may help to have a magnifier at hand.)

In figure 1 we take a first look at some of the reshaping. The gray area is the bounding box, the white shape is the new font, the outline is the old one.

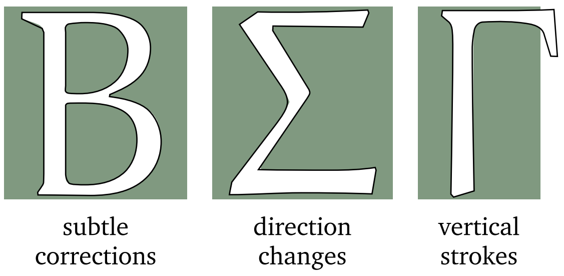


Figure 1. New Euler Roman Medium (a)

In figure 2 we see more drastic changes: shortened strokes. The bounding box is kept unchanged since we made it an initial goal for the new shapes to work well with the existing metric files; that way, New Euler would be a drop-in replacement for the existing fonts and could be used with no fear of changing line breaks.

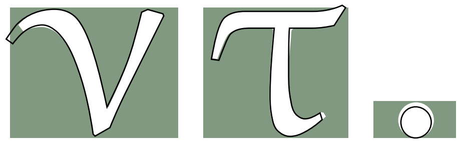


Figure 2. New Euler Roman Medium (b)

As with Palatino Nova and Optima Nova, Hermann did not hesitate to go even further than this. Figure 3 demonstrates this clearly. A nice side effect of harmonizing the font is that we can now use Euler for running text, although the text font is not yet released (due to too many holes in the usual text encoding vectors).

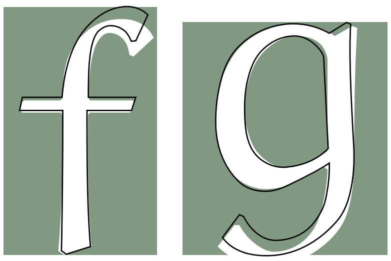


Figure 3. New Euler Roman Medium (c)

Some of the changes result directly from looking at the fonts in a larger size (see figure 4). The redesign started by printing the outlines of the fonts at sizes up to 12cm but finally Hermann decided to focus on the 6cm variant. The corrected outlines were mailed, faxed and/or presented in person. Many such corrections concerned the way corners are cut off. In that respect some of the original characters didn't qualify as Euler at all, for instance $<$ symbols, but by cutting some corners and adapting the strokes they became eulerized.

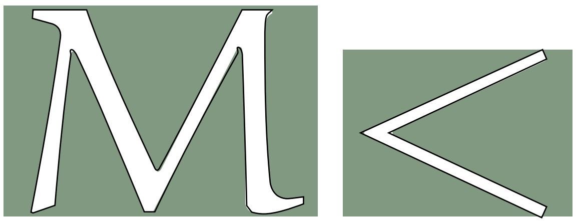


Figure 4. New Euler Roman Medium (d)

When the discussions about reshaping started, the changes mostly concerned small corrections and descenders, but once we had the proper work-cycle in place Hermann went a bit further. Of course the descenders have been lowered too, as is demonstrated in Figure 5.

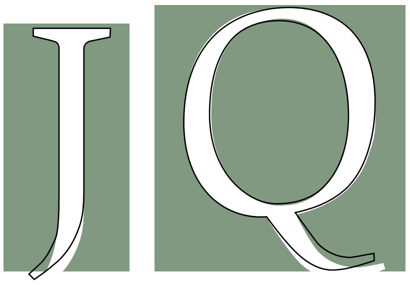


Figure 5. New Euler Roman Medium (e)

As usual, TEX math fonts have interesting ways of combining characters in fonts and so we have old style digits in the Fraktur font. The elegance of New Euler is well demonstrated by these numerals (see figure 6).

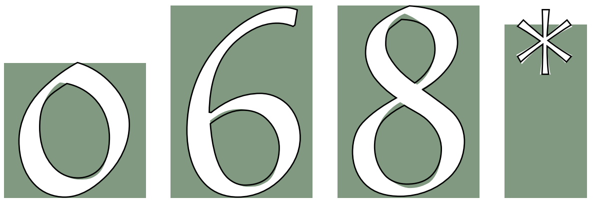


Figure 6. New Euler Fraktur Medium

Most characters have been changed, some much more than others. In the symbol font, the aleph now better matches the rest (it was rather fat) and the script L is less upright (see figure 7).

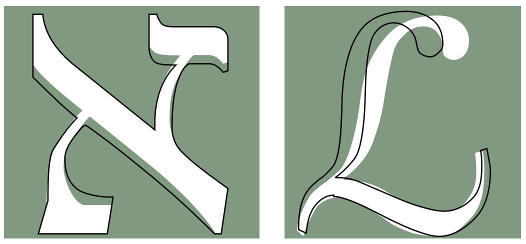


Figure 7. New Euler Symbol Medium

As intended, New Euler is metric compatible with Old Euler, and of course the smaller sizes and the bold variants have been adapted too. By the end of 2007 all the medium variants at 10pt were done, and Taco had to go into overdrive. We were quite lucky that he has mastered FontForge so well (figure ??), and so we could start 2008 with a complete set of fonts.

The fonts were presented to Don Knuth on January 10, 2008 on an eight-page leporello hand-made by Willy Egger, with each page showing one of the aspects of the reshaping, all of which kept us pretty busy during the holidays (figure 8).

Does the project end here? No, this is just the first stage. Hermann is willing to participate in extending the Euler fonts with the Unicode math characters that make sense.

An important aspect of the project is to get the old fonts replaced by the new ones. We're very happy that Barbara Beeton has managed to convince the AMS folks to accept the new font as a formal substitute for the existing ones. And of course, the TEX distribution wizard Karl Berry will take care of getting the fonts in the right places and rooting out traces of old ones.

Figure 8. Presenting New Euler

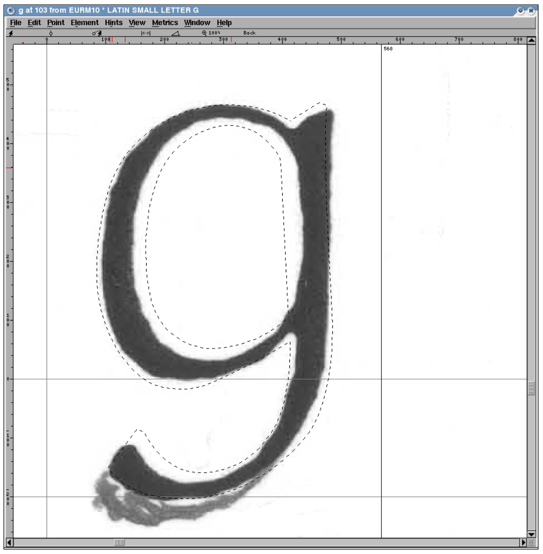

g at 103 from EURM10 * LATIN SMALL LETTER G
File Edit Point Element Hints View Metrics Window Help

Figure 9. Editing Euler in FontForge

For quite some time Don Knuth has been asking users to get rid of the old Computer Modern delta, so in closing let's quote him from his web site on behalf of Hermann:

> If you see that your system produces the symbol δ instead of δ for the Greek lowercase delta, you should tell your system administrator immediately to upgrade your obsolete version of the Euler fonts.

And don't tell us that you don't see the difference. And, as you may expect, this quote was typeset in Euler Text.

Hans Hagen, Taco Hoekwater, Volker RW Schaa

Blocks and Arrows with MetaPost

Abstract
Typesetting of blocks and arrows in ConTEXt with MetaPost.

Keywords
MetaPost, ConTeXt, color, drawing, block, arrow, label.

Introduction

There were a number of reasons for the development of my own package for drawing blocks and arrows. The first is the past experience with the use of LaTEX for these. Its pictures always have a touch of imperfection caused by the restricted set of line and arrow directions available. Using home made sets of arrowheads and lines in more directions than in the LaTEX-fonts, alleviates this problem somewhat but not enough. Drawing with MetaFont gives a lot more satisfaction, albeit with more effort. So it was decided to develop macros for block and arrow-drawing with a sufficient level of sophistication.

Blocks

As an appetizer figure 1 shows a catalogue of the shapes preprogrammed in the package. After the presentation of the drawing API it will be told how one can easily add its own shapes.

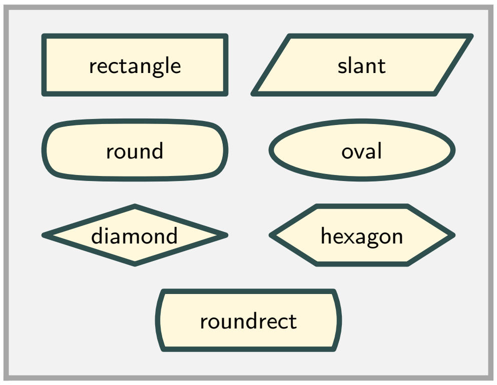


Figure 1. Block catalogue

All shapes are drawn from the path of a unit figure ($x = 0 \cdots 1, y = 0 \cdots 1$) and should be sized by setting their width and height. The following shapes are present:

- *rectangle* – unit square scaled in the x and the y-direction;
- *slant* – square with a parametrized horizontal shearing;
- *round* – parametrized superellipse;
- *oval* – basically a unit circle;
- *diamond* – a lozenge;
- *hexagon* – six corners regularly spaced;
- *roundrect* – square with parametrically curved vertical sides.

The shapes in figure 1 are placed with the following macro call, where *rectangle*, *slant*, etc. are substituted for *shape*:

```
Block.shape (center, width, height, labeltext);
```

The *rectangle* is the default shape when calling `Block` without the shape modifier. The parameter `center` gives the distance from the origin over which the center of the shape is displaced. The parameters `width` and `height` scale the shape to the dimensions required. Finally `labeltext` is a *string* typeset in the center.[1]

Some of these shapes are parametrized; in the macro call above this parameter is given default value 0. Figure 2 shows the effect of various parameter values on the *slant* and the *roundrect* shapes. These figures are drawn with a more elaborate version of the previous macro:

```
VarBlock.shape (param, rotation, center, width,
                height, labeltext, outline);
```

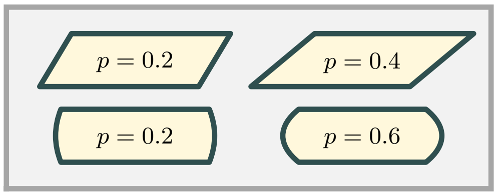


Figure 2. Parametrized shapes

There are some extra parameters here. The shape parameter is `param`, the `rotation` is an angle (degrees, counterclockwise) rotating the shape around its center before being shifted in place, and `outline` is a boolean governing the drawing of the frame around the shape. The programmed default is taken for `param` = 0, but as we will see this can be easily changed by altering the shape definition. Figure 3 illustrates the rotation of a shape with various angles.

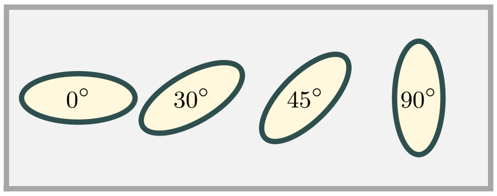


Figure 3. Rotated shapes

The `outline` parameter makes it possible to omit the framelines, leaving the colored shape only. Note in figure 4 the small difference in size between the two shapes, caused by the centering of the linedrawing on the boundary of the shape.[2] In order to facilitate drawing of shapes without outline one can use macro `OBlock` in the same manner as Block:[3]

```
OBlock.shape(center, width, height, labeltext);
```

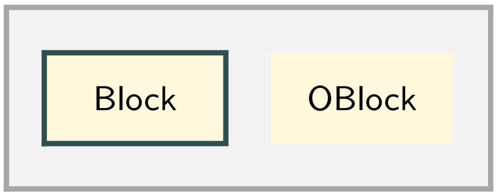


Figure 4. Outline parameter

The coloring of pictures is based on a set of four different colors; the management of those is explained in section “Color management”. The distinct colors are:

1. the color for linedrawing
2. the color for the interior of shapes
3. the color for the labeltext
4. the color for the general background

In the example of figure 5 two sets of colors are used.

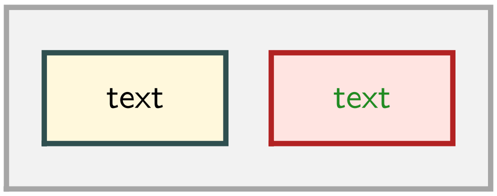


Figure 5. Different coloring

The rectangular frames around the figures are drawn by a simplified shape drawer. There are three macros:

```
Frame (width, height, gap);
Framed (width, height, gap, framepen);
Framed (width, height, gap, framepen) modifier;
```

The first macro produces a rectangular block of size $(gap + width + gap) \times (gap + height + gap)$ filled with the general backgroundcolor. The second one uses the *pen* parameter `framepen` to surround the block with framelines in the color for linedrawing. The third gives the possibility to individually modify the surrounding framelines in linestyle and color. The gap is an enlargement on all sides of the block; it was introduced in order to prevent lines drawn at the borders of the drawing area from spilling partly over into the surroundings. The resulting frame is placed at position $(-gap, -gap)$.

As is the case with `Framed` one can have a modifier on the framelines following calls to `Block`, `OBlock` and `VarBlock` too. The right side of figure 6 has been done with modifier `dashed evenly withcolor 0.7yellow`.[4]

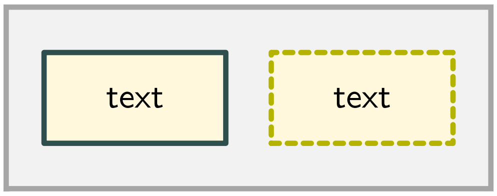


Figure 6. Trailing modifier

Defining shapes

It is quite easy to define shapes yourself and call them up with `Block.shape`. Lets have a look at how shapes are done. Shape drawing is effected by macro `Form` that expects to receive a *path* of the shape in one of its parameters; it rotates and shifts the path as required, fills the resulting path, places the labeltext and possibly surrounds it with framelines. All this is done in the same way for all shapes.

The path of the shape is constructed by helper macro `varblock@#`. For example `Block.oval` uses a predefined *string* `blockf.oval` which contains the path expression for a circle fitting within the square bounded by $x = 0 \cdots 1, y = 0 \cdots 1$. That expression, possibly parametrized with the shape parameter, is scanned and processed, the resulting path being scaled to its proper width and height. Afterwards the string `blockl.oval` is scanned for any postprocessing action that might be needed. The *round* shape (actually a superellipse) needs such postprocessing; for the others this is just a noop i.e. an empty string.

All that is needed therefore is the definition of these two strings. As an example a fourpointed star is developed; its four points ending at the midpoints of the sides of the unit rectangle, the intersections between them parametrically placed on the diagonals (see figure 7).

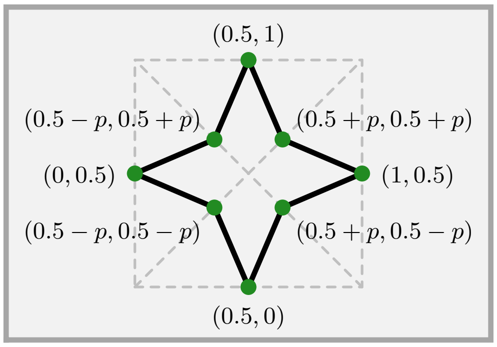


Figure 7. Plan of fourpointed star

In the code for the fourpointed star below note the *numeric* variable `p_`. The expansion of `VarBlock` assigns `param` to this variable.[5]

```
% Define path for fourstar.
vardef unitfourstar =
       (0.5,0)
       -- (0.5+p_,0.5-p_) -- (1,0.5)
       -- (0.5+p_,0.5+p_) -- (0.5,1)
       -- (0.5-p_,0.5+p_) -- (0,0.5)
       -- (0.5-p_,0.5-p_) -- cycle
enddef;
% Define scan strings for fourstar.
string blockf.fourstar, blockl.fourstar;
blockf.fourstar :=
    "if p_ = 0: p_ := 0.12 fi; unitfourstar";
blockl.fourstar := "";
```

This code is used in the placement of the two stars in figure 8 with:

```
Block.fourstar (.., "star");
VarBlock.fourstar (0.05, 45, .. , "", true);
```

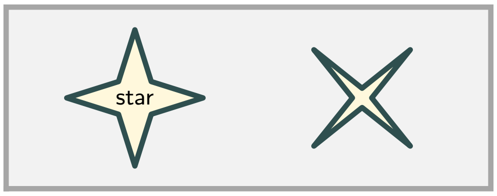


Figure 8. Fourpointed stars

Labels

Besides the standard `label` macro in *plain* one can use the `Label` macro instead. The differences are subtle, the most important one being the typesetting of the labeltext. In `Label` the text is placed with ConTEXt-macro `textext` into a `hbox` whose depth is forced to zero. The reason for this is seen in the top boxes of figure 9, which are typeset with the original `label` macro: the baselines in the two boxes differ because of their different depths. At the bottom the `Label` macro has put both texts neatly on the same baseline. The effect is clearly seen in the position of the e's and d's with respect to the dashed line. A second refinement is the fact that the labeltext can be both in the form of a*picture* or a *string*. Since the `Label` macro is used in the typesetting of shapes, this applies there too.

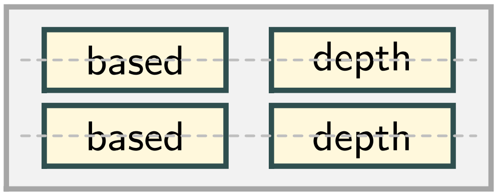


Figure 9. Label placement

Arrows

Uneasiness grew over the shape of the arrows provided by the *plain* macros. This is illustrated in figure 10. The shape of the arrowheads on the left could be called somewhat more refined than those on the right. Moreover the *plain* macros distort the arrowhead at the end of a curved line, bending their flanks with the curve. It is, of course, a matter of taste to find this appalling. But more objectively stated, *plain* makes it impossible to consistently draw arrowheads in the same shape. A suitable alternative has given by David Salomon and is implemented here.[6]

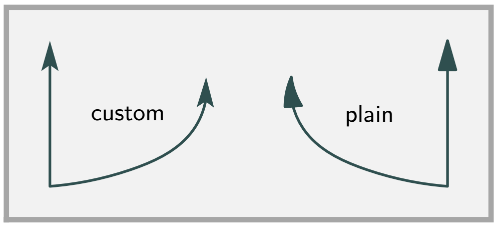


Figure 10. Arrow shapes compared

Figure 11 shows the geometry of the arrowhead. There are several parameters that can be varied:

- l is the total length of the arrowhead, in the figure the distance between the two red bars;
- r is the ratio that determines the distance from the tip to the point where the arrowhead is affixed to the incoming line (absolute length is $r \times l$);
- α is the opening angle at the tip and determines the distance between the wingtips;
- β gives the curvature of the flanks (here 10°), the straight gray line is the one drawn for $\beta = 0^\circ$;
- γ gives the curvature of the tailtips (here 2°), the straight gray line is the one drawn for $\gamma = 0^\circ$.

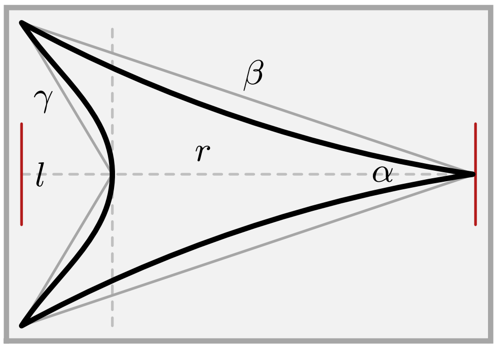


Figure 11. Arrowhead

With this geometry available, one can build many different arrowheads; figure 12 presents some. As can be seen, the heads may or may not have an outline. Be aware that not all parameter value combinations lead to pleasing looking shapes. Arrowshapes are defined with the following macro:

```
defineArrow (l,r,alfa,beta,gamma,outline) name;
```

The first five parameters are those from figure 11. The `outline` parameter is a boolean designating the drawing of an open arrowhead when `true`. In an open arrowhead interior and outline are filled with the colors for filling and linedrawing, respectively; `false` results in a solid head in the linedrawing color. It is necessary to select pen and colorset before the definition is executed, because these are used at that point. The parameter `name` is a user chosen unique designation by which the arrowhead later will be retrieved for placement in the drawing. Nota bene: this parameter must be a *string*. The heads thus defined keep their coloring even when a different colorset is selected later.

The arrows so defined are placed with the following macros for single and doublesided arrows, respectively. Here too the `name` must be a *string*:

```
drawArrow (name) path_expression;
drawdblArrow (name) path_expression;
```

The lines to the arrowheads adapt to the linecolor in effect at definition time, but this can be overridden with a modifier for both linestyle as well as color (see the rightmost arrow in figure 12).

The first of the next three macros defines a solid arrowhead for default usage.[7] The next two draw these default arrows, respectively single and doubleheaded; they are illustrated in figure 13. The drawing is fully in the current linedrawing color. However a modifier may be applied to change linestyle and/or drawing

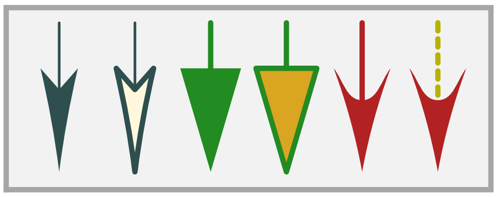


Figure 12. Arrowhead variations

color, as in the two arrows on the right. Changing the color for subsequent drawing can also be effected by changing the color for linedrawing.

```
defineDefaultArrow (l, r, alfa, beta, gamma);
arrowline path_expression;
dblarrowline path_expression;
```

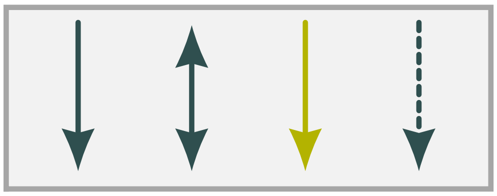


Figure 13. Default arrow drawing

Color management

The coloring scheme introduced in section “Blocks”, consists of four colors used in linedrawing, filling the interior of shapes, drawing labeltext and the general background. The color for filling may be given an alpha mode and a factor for transparency. These colors can all be set individually with the four macros:

```
setFrameColor color;
setFillColor color;
setTextColor color;
setBackgroundColor color;
```

In modifiers they can be applied with a shortcut for `withcolor color` by using:

```
withFrameColor
withFillColor
withTextColor
withBackgroundColor
```

The alpha mode and fill can be changed through:

```
setFillMode mode;
setFillAlpha factor;
```

Sometimes one needs a different set of colors just momentarily, then the following macros come in handy. Their names speak for themselves. Note that saving cannot be nested, resaving causes the previous save to be lost; therefore do not forget to restore in case.

```
saveColors;
restoreColors;
```

There exist the following macros for the definition of a set of colors:

```
defineColors  (line,fill,text,background) name;
defineColorsTransparent
  (line,fill,text,background,mode,factor) name;
defineCurrentColors name;
defineDefaultColors
  (line, fill, text, background);
defineDefaultColorsTransparent
  (line, fill, text, background, mode, factor);
setDefaultColors;
```

As was done for arrow shapes, `name` defines a designator with which to call up the thus named set of colors. Nota bene: this parameter must be a *string*. If the transparency parameters are not explicitely given, they assume default values of 1 for mode and factor, leading to completely opaque colors. The third macro eases entering a definition for the colors currently set, the fourth and fifth define a default that can be called up by the last one. Colorsets are set by the next two macros; here too `name` must be a *string*. The second one below is just a set followed by a save.

```
setColors name;
setSaveColors name;
```

Notes

1. This is not completely true; in section 4 it is mentioned that a *picture* is allowed also.
2. In case one wants to combine shapes with and without outlines, the individual widths and heights can be adjusted by the size of the pen in the respective dimension. The macros `penX` and `penY` provide these for the current pen.
3. For those inclined to perfection: compare the rectangles in figures 4 and 6 for the use of a square pen in the former.
4. Be aware that a dashed modifier applies to a `pencircle`, but has no effect with `pensquare`.
5. It arises because the formal parameters of the macro definition cannot be used in the `blockf` string, which is processed by `scantokens`. Also note the use of `vardef` in the path definition macro, `def` will not work here.
6. David Salomon, Arrows for Technical Drawings, TUGboat, Volume 13 (1992), No.2, p. 146–149.
7. Be aware that the implementation of the definition code thus not guarantee recycling of the memory for redefined entries. One should there not redefine too enthousiastically.

Hans van der Meer
hansm@science.uva.nl