XeTeX Live

Jonathan Kew
SIL International
Horsleys Green
High Wycombe
Bucks HP14 3XL, England
jonathan_kew (at) sil dot org

1 XeTeX in TeX Live

The release of TeX Live 2007 marked a milestone for the XeTeX project, as the first major TeX distribution to include XeTeX (version 0.996) as an integral part. Prior to this, XeTeX was a tool that could be added to a TeX setup, but version and configuration differences meant that it was difficult to ensure smooth integration in all cases, and it was only available for users who specifically chose to seek it out and install it. (One exception to this is the MacTeX package, which has included XeTeX for the past year or so, but this was just one distribution on one platform.) Integration in TeX Live, in contrast, provides near-universal availability and a more standardized configuration, which should simplify setup, use and support.

Special thanks to Karl Berry for his encouragement and support through this process, and to all the TeX Live builders and testers on various platforms who helped to make this possible.

1.1 Key features

The two most significant features of XeTeX as found in TeX Live remain the same as they have been since its first appearance: support for the use of the host operating system's fonts (PostScript, TrueType, or OpenType) with no TeX-specific setup, and including layout features defined in the fonts; and extensive support for Unicode, including complex Asian and other scripts. With this release, users on all platforms have the option of using the same OpenType fonts in TeX documents as in mainstream GUI applications, including access to all the rich typographic features found in modern fonts.

As an example of the simplicity XeTeX brings to font usage, consider the present article. This is written using the `ltugproc` class. Running this in XeLaTeX, the lines:

```
\usepackage{fontspec}
\setmainfont[Mapping=tex-text]
            {Adobe Garamond Pro}
\setmonofont[Scale=MatchLowercase]
            {Andale Mono WT J}
```

in the preamble are sufficient to set the typefaces throughout the document. These fonts were installed by simply dropping the `.otf` or `.ttf` files in the computer's `Fonts` folder; no `.tfm`, `.fd`, `.sty`, `.map`, or other TeX-related files had to be created or installed.

Release 0.996 of XeTeX also provides some enhancements over earlier, pre-TeX Live versions. In particular, there are new primitives for low-level access to glyph information (useful during font development and testing); some preliminary support for the use of OpenType math fonts (such as the Cambria Math font shipped with MS Office 2007); and a variety of bug fixes.

1.2 Hyphenation setup

A long-standing problem with integrating XeTeX has been the variety of hyphenation patterns for various languages, which are written using a variety of character encodings and various ways to represent those encodings in 7-bit or 8-bit files. Because XeTeX interprets 8-bit text files as Unicode (UTF-8) by default, many old hyphenation files cannot be read as-is. This in turn meant that the XeLaTeX format could fail to build, depending on the user's language configuration.

Older releases of XeTeX made some attempt to address this by including modified versions of some of the hyphenation files from teTeX, adapted to load correctly as Unicode patterns. However, ensuring that these were installed in the right place for XeTeX to find them (without affecting other engines or replacing standard files) was problematic.

In TeX Live 2007, this situation has been addressed by modifying the `language.dat` file so that hyphenation files are loaded via "wrappers" (except for those that are simple ASCII files, which are already Unicode-compatible). The wrapper files, provided in TeX Live in `texmf-dist/tex/generic/xu-hyphen`, test whether the format is being built by XeTeX, and if so they redefine the input encoding and/or `\catcodes`, active character definitions, etc., so that the patterns will be loaded as Unicode data. Figure 1 shows an example of such a wrapper file; in this case, the German vowels with umlauts and the ß character need Unicode-compliant definitions, in place of those found in the original hyphenation file. The precise details vary, of course, depending on the structure and encoding

Note: This article is based on the author's presentations at both the EuroBachoTeX 2007 and TUG 2007 conferences, but is printed in a single *Proceedings* issue to avoid duplication.

```
% xu-dehyphn.tex
% Wrapper for XeTeX to read dehyphn.tex
% Jonathan Kew, 2006-08-17
% Public domain
\begingroup
\expandafter\ifx\csname XeTeXrevision\endcsname\relax
\else
  \catcode`\?=7
  % Define the accent macro " in such a way that it
  % expands to single letters in Unicode
  \catcode`\"=13
  \def"#1{\ifx#1a??e4\else \ifx#1o??f6\else \ifx#1u??fc\else
      \errmessage{Hyphenation pattern file corrupted!}%
    \fi\fi\fi}
  %   - patterns with umlauts are ok
  \def\n#1{#1}
  %   - define \3 to be character "OODF (\ss in Unicode)
  \def\3{??df}
  %   - define \9 to throw an error
  \def\9{\errmessage{Hyphenation pattern file corrupted!}}
  %   - duplicated patterns to support font encoding OT1 are not wanted
  \def\c#1{}
  %
  \let\PATTERNS=\patterns
  \def\patterns{% at the \patterns command in dehyphn.tex...
    \endgroup % end group containing local definitions from dehyphn
    \begingroup % and start our own (to match \endgroup in dehyphn)
    \PATTERNS % and then load the real patterns
  }
\fi
\input dehyphn.tex
\endgroup
\endinput
```

Figure 1: `xu-dehyphn.tex`, a typical hyphenation wrapper file from the TEX Live setup

of the pattern file being loaded, but similar techniques can generally be used.

In the longer term, reorganization and standardization of the hyphenation files, perhaps co-ordinated with work in OpenOffice.org (which uses a very similar hyphenation algorithm) would be a useful project. However, this will require not only a good understanding of the language and encoding issues, but also interaction with license holders or maintainers of all the existing patterns. Meanwhile, the current setup with `xu-` wrappers has proved to be a workable interim solution.

1.3 Package configuration

Another common problem for XeTeX users in the past has been that some popular LaTeX packages (e.g., `graphics`, `color`, `geometry`, `crop`, `hyperref`, and others) depend on knowing the intended output driver (direct PDF generation with pdfTeX, `dvips`, `dvipdfm`, etc.) in order to use the correct implementation-specific methods to control the output. Many such packages attempt to detect the TeX engine in use and automatically choose the appropriate driver. However, with XeTeX being a new engine, existing packages were unaware of it.

This situation is improving, as some major packages have added a test for XeTeX and now choose the appropriate driver options. For others, including important cases like `geometry` and `crop`, TL2007 includes configuration files in the `xelatex` subtree that provide the proper setup. In most cases, therefore, users should find that the packages work transparently in XeTeX just as with other TeX engines and drivers.

One important package that did not work transparently with XeTeX in the TL2007 release is `pgf`; however, since the release in February, `pgf` has also been updated so that it recognizes the XeTeX engine automatically.

1.4 The ArabXeTeX package

A new package by François Charette provides an ArabTeX-like interface for typesetting languages in Arabic script, using standard Unicode-based fonts. As shown in figure 2, this supports both literal Unicode input of Arabic text, and ArabTeX transliterations, and can work with any OpenType font, including complex calligraphic styles such as Nastaliq script. This package was created after the current TeX Live release, but can be obtained from CTAN and works with the existing XeTeX version.

```
% preamble...
\usepackage{arabxetex}
\newfontfamily\arabicfont
     [Script=Arabic]{Scheherazade}
\newfontfamily\urdufont
    [Script=Arabic,Scale=0.75]{Noori Nastaliq MT}
% body...
\begin{arab}[fullvoc]
mina 'l-qur'Ani 'l-karImi,
   sUraTu 'l-ssajdaTi 15--16:
% ...etc...
\end{arab}
\begin{urdu}[voc]
اَنبیاءِ بنیاِسرائِیل وچّوں حضرت یُونسؑ
% ...etc...
```

Result:

مِنَ ٱلْقُرآنِ ٱلْكَرِيمِ، سُورَةُ ٱلسَّجْدَةِ ١٥–١٦:

إِنَّمَا يُؤْمِنُ بِءَايـٰتِنَا ٱلَّذِينَ إِذَا ذُكِّرُوا بِهَا خَرُّوا سُجَّداً وَسَبَّحُوا بِحَمْدِ رَبِّهِمْ وَهُمْ لَا يَسْتَكْبِرُونَ ۩ ﴿١٥﴾ تَتَجَافَىٰ جُنُوبُهُمْ عَنِ ٱلْمَضَاجِعِ يَدْعُونَ رَبَّهُمْ خَوْفاً وَطَمَعاً وَمِمَّا رَزَقْنٰهُمْ يُنْفِقُونَ ﴿١٦﴾

اَنبیاءِ بنی اِسرائیل وِچّوں حضرت یُونسؑ ہِک جلیل القدر نبی ہن۔ آپ دا دَورِنبوّت ۷۸۲ /۷۸۱ق م کنُوں ۷۵۳ق م تک ہے۔ اُوں ویلے مُملکتِ بنی اِسرائیل دا بادشاہ یُربعام ثانی ہا۔ آپ اِیں مُملکت وِچ ناصرت دے نال "جات حفر" دے مقام تے رہندے ہَن۔ اَج کَل اِیں جاہ کُوں "خربۃ الزُّرّع" آہدِن۔ ایہ مقام جھیلِ جلیل دے مغربی پاسے بارھاں میل اتیں ناصرۃ دے شُمال مشرق پاسے ترائے میل دے فاصلے تے ہے۔ اِتُّھوں دے کھنڈرات دی کُھدائی دے دَوران پتّا لگّے جو ایہ مقام کانسی دے آخری دَور تقرِیباً ۱۵۵۰ق م کنُوں ۱۲۰۰ق م تک آباد ہا۔ وَلا ڈُوجھی دفعہ ایہ تقرِیباً ۱۲۰۰ق م کنُوں ۶۰۰ق م تک آباد رِیہا۔ حضرت یُونسؑ ہیں ڈُوجھے دَور وِچ ہَن۔ اِنہیں کھنڈرات کنُوں ذرا شُمالی پاسے ہِک وستی جیندا ناں ہے "مشہد"۔ روایت دے مُطابق حضرت یُونسؑ دا مزار اِتھائیں ہے۔

Figure 2: Examples of ArabXƎTEX input and typeset output

2 Beyond TL2007 and XƎTEX 0.996

In parallel with the integration of XƎTEX 0.996 into TEX Live, there has been continuing development of the next version of XƎTEX itself and the associated drivers and support files. Release 0.997 (preliminary code is in the Subversion source repository at the time of writing) will include several new and enhanced features, a few of which are described here.

2.1 PSTricks graphics

One of the limitations of XƎTEX has been that it natively generates `.xdv` or "extended DVI" output, which needs to be converted to PDF by a special XƎTEX-specific output driver. This excludes the use of the `dvips`+Ghostscript output path, and therefore also prevents the use of packages that rely on writing PostScript `\special` commands that Ghostscript or a PostScript printer will interpret.

The most important such package, judging by discussion on the mailing lists, is probably PSTricks, which is widely used for special drawing and graphic effects. Thanks to recent work by Miyata Shigeru, the `xdvipdfmx` driver used with XƎTEX has been extended to support most PSTricks features (with a few exceptions), and therefore standard PSTricks pictures, plots, etc., can be used in XƎTEX. This is achieved by extracting the PostScript code and running Ghostscript (or another process, according to the driver's `.cfg` file) to convert this to PDF which can then be embedded in the document, as illustrated in figure 3. While this technique is currently quite slow, it does at least permit the use of such graphics. However, users may find that other graphics packages such as the PGF-based Ti*k*Z provide better performance in many cases.

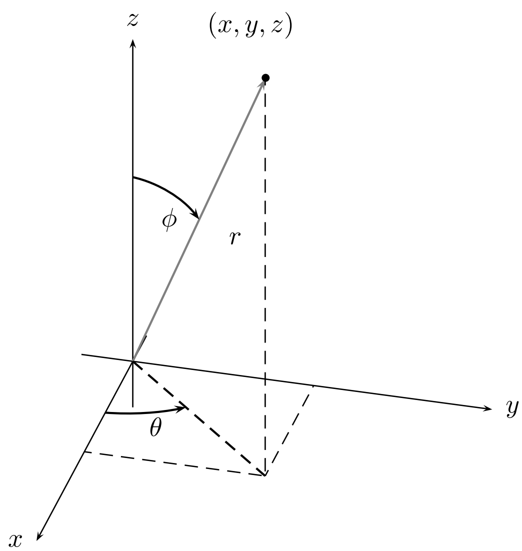

Figure 3: Example of a PSTricks plot embedded in a XƎLATEX document (from `http://tug.org/PSTricks/main.cgi?file=pst-plot/3D/examples`)

2.2 Unicode math extensions

New in XƎTEX 0.996, and more complete in 0.997, is support for use of the full range of Unicode math characters, including the styled math alphabets in Plane 1 as well as the large number of mathematical symbols. TEX's `\mathcode`, `\delcode` and related tables have been enlarged, and the number of math families is increased from 16 to 256. A small example of the use of Unicode characters in math mode is shown in figure 4; work is in progress to design and implement a LATEX package to provide extensive and well-integrated support, building on the primitive facilities now available in the engine.

TEX's math codes contain three distinct components, representing the character class (ordinary character, large operator, binary operator, relation, etc.), the math family to be used, and the actual character code. TEX compresses this information into a single 16-bit value, with 3 bits for the class, 4 for the family, and 8 for the character code, normally expressed as 4 hex digits (see *The TEXbook*, p. 154). XƎTEX packs a 3-bit class, 8-bit family, and 21-bit Unicode value into a single 32-bit code, but as the example in figure 4 shows, it allows the components to be specified separately for clarity as they no longer map neatly onto individual hex digits.

```
% set up Cambria Math for roman, symbol and extension families
\font\1="Cambria Math:script=math" at 10pt
\font\2="Cambria Math:script=math;+ssty=0" at 7pt
\font\3="Cambria Math:script=math;+ssty=1" at 5pt
\textfont0=\1 \scriptfont0=\2 \scriptscriptfont0=\3
\textfont2=\1 \scriptfont2=\2 \scriptscriptfont2=\3
\textfont3=\1 \scriptfont3=\2 \scriptscriptfont3=\3

% use Cambria Math with italic mapping for family 1
\font\1="Cambria Math:script=math;mapping=math-italic" at 10pt
\font\2="Cambria Math:script=math;mapping=math-italic;+ssty=0" at 7pt
\font\3="Cambria Math:script=math;mapping=math-italic;+ssty=1" at 5pt
\textfont1=\1 \scriptfont1=\2 \scriptscriptfont1=\3

% set mathcodes (many are predefined in xetex.fmt)
\XeTeXmathcode`\-="2 "2 "2212 % minus sign
\XeTeXmathcode`\∑="1 "2 `∑ % summation

% some control sequences...
\XeTeXmathchardef\sum="1 "2 `∑     \XeTeXmathchardef\prod="1 "2 `∏
\XeTeXmathchardef\intop="1 "2 `∫   \XeTeXmathchardef\infty="1 "2 `∞
\XeTeXmathchardef\geq="3 "2 `≥     \XeTeXmathchardef\leq="3 "2 `≤
\XeTeXmathchardef\pi="7 "1 `π

% using Unicode characters in math
$$ f(x) = a_0 + ∑^∞_{n=1} \left( a_n\cos{nπx\over L} + b_n\sin{nπx\over L} \right) $$
```

Result, using an OpenType math font:

$$f(x) = a_0 + \sum_{n=1}^{\infty} \left(a_n \cos \frac{n\pi x}{L} + b_n \sin \frac{n\pi x}{L} \right)$$

Figure 4: Defining and using Unicode math characters

When using a complete OpenType math font such as Cambria Math, it may be necessary to load the font several times with different character mappings and OpenType features.

2.3 Inter-character token insertion

A new feature in XeTeX version 0.997 is the ability to insert arbitrary token lists in between normal text characters, without complex macro programming. This is designed primarily to support requirements of Japanese and Chinese typography, where special spacing controls are needed in certain cases such as between ideographs and adjacent punctuation characters.

To support this feature, each character has a "class" known as `\XeTeXcharclass`, a bit like an extra `\catcode`, but ignored by normal TeX operations. But whenever two printable text characters occur next to each other, XeTeX will check their class values, and if a token list has been defined for this class pair it will be inserted between the characters. Such a token list may contain arbitrary TeX material, although the most useful possibilities are probably various forms of `\skip` and `\penalty` (to control spacing and breaking), and font changes (making it possible to automatically switch fonts for different scripts within Unicode text, without requiring embedded markup).

For example, the default `xetex` and `xelatex` formats initialize most `\XeTeXcharclass` values to zero, but assign all the CJK ideographs to class 1. We can take advantage of this to allow Chinese characters to be included in running text without additional markup, even though the default body font does not support them; a simple example is shown in figure 5. While this technique is not a universal substitute for proper language and font markup in the source document, it can greatly simplify the author's task in some mixed-script situations.

2.4 Graphite font support

The initial version of XeTeX, on Mac OS X only, supported special font features such as contextual swashes, ligatures, alternate glyphs, etc., by means of Apple's AAT font technology. Later, support for OpenType font features was added, based on the ICU layout library; this enabled XeTeX to provide complex font support across multiple platforms.

A third font layout technology, designed to support the requirements of non-Latin scripts, minority languages, and scripts not yet in Unicode, is SIL's Graphite system (`http://scripts.sil.org/RenderingGraphite`).

```
\newfontfamily\zhfam{STKaiti}
  % define a font for Chinese

% xelatex has CJK ideographs assigned to class 1
% and Latin (etc) in class 0 by default
\XeTeXinterchartoks 0 1 = {\begingroup\zhfam}
  % switch to \zhfam when we find a Chinese char
\XeTeXinterchartoks 1 0 = {\endgroup}
  % and revert at the end of a Chinese run

% class 255 is a special "end of string" marker,
% so we need to switch here as well
\XeTeXinterchartoks 255 1 = {\begingroup\zhfam}
\XeTeXinterchartoks 1 255 = {\endgroup}

The Chinese word 你好 means ‘hello’.
```

Result: The Chinese word 你好 means ‘hello’.

Figure 5: Using inter-character token insertion to mix scripts and fonts without in-line markup

```
\font\myfnt="Padauk/GR" at 7.5pt
\XeTeXlinebreaklocale "G"
\XeTeXlinebreakskip=0pt plus 1pt
\XeTeXlinebreakpenalty=10
\baselineskip=14pt

\myfnt ကျွန်ုပ်တို့၏ ပျော်ရွှင်မှု၊ သာယာ ဝ ပြောမှုနှင့် အောင်မြင်မှုတို့သည်...
```

Result: ကျွန်ုပ်တို့၏ ပျော်ရွှင်မှု၊ သာယာ ဝ ပြောမှုနှင့် အောင်မြင်မှုတို့သည် ကျွန်ုပ်တို့၏ ကျန်းမာခြင်းအပေါ်တွင် အများကြီး မှီခိုနေပါသည်။ ပညာတတ်ရန်၊ ကြွယ်ဝချမ်းသာရန်နှင့် ကြိုးပမ်း လုပ်ဆောင်မှုအားလုံး အောင်မြင်စေရန်အတွက် ကျန်းမာရေးသည် အထူးပင် အရေးကြီးပါသည်။ ကျန်းမာရေးမပြည့်စုံလျှင် ကျွန်ုပ်တို့၏ ပညာရေး၊ စီးပွားရေး မြှင့်တင်မှုလုပ်ငန်းများ လုပ်နိုင်လိမ့်မည် မဟုတ်ပါ။ သို့ဖြစ်၍ အစဉ်သဖြင့် ကျန်းမာနေရန် ကျွန်ုပ်တို့ ကြိုးစားကြရပါမည်။

Figure 6: Burmese script rendered using a Graphite font (compare the source text, rendered without Graphite technology, to the resulting output)

Graphite provides a level of glyph layout control within the font that goes beyond either AAT or OpenType, making it possible for font developers to provide more flexible and accurate rendering of multiple diacritics and other characters that interact with their neighbors in complex ways.

A small example of Graphite text rendering is shown in figure 6, where a Graphite font is used to provide correct rendering of Burmese script. The font also includes custom line-breaking rules, which XƎTEX uses to find valid breaks within the text, even where no spaces are present.

Another script being typeset using Graphite is N'Ko, a writing system from West Africa. This is a cursive writing system, written from right to left, but unrelated to Arabic script. It has recently been standardized in Unicode (version 5.0), but is not yet widely implemented; I have not yet seen a working OpenType implementation, for example. But because Graphite allows the font developer full

```
\font\x="N'ko Kankan/GR:rtl" at 8.5pt
\noindent\beginR
\x [illegible] % ...etc...
```

Result: [illegible]

Figure 7: The N'Ko script rendered using a Graphite font

control of the rendering behavior, without a script-specific "shaping engine" (as OpenType requires), N'Ko users already have a full-featured typesetting solution that handles the script. Figure 7 shows a short sample of N'Ko input (using default glyphs, without contextual rendering) and the corresponding properly-rendered text.

2.5 Implementation details: pool file and formats

A couple of recent changes to the internal implementation of XƎTEX may be of interest, and could be considered by other TEX systems as well. One (following METAPOST's lead) is that there is no longer a separate `.pool` file used to initialize the program's strings; instead, the strings are compiled into the program file itself. This simplifies installation and maintenance, and removes the possibility of a version mismatch. It is particularly relevant in the multi-platform environment of TEX Live, as it means that a new XƎTEX binary could be provided for some platforms ahead of others, without the problem of deciding which version of the `.pool` file should be included in the `texmf` tree.

With more complete Unicode math support, various per-character code tables have been extended to support the full Unicode range of around one million possible character codes. (Formerly, only the 64K characters of the Basic Multilingual Plane had individual `\catcode`, `\lccode`, `\mathcode`, etc. values.) As the default formats initialize these tables based on the Unicode standard, which currently defines about 100,000 characters, the resulting `.fmt` files became considerably larger. To alleviate this, the format file reading and writing routines now use the well-known `zlib` compression library; as `.fmt` files are typically quite compressible, this leads to large space savings, and the reduction in disk I/O compensates for much of the overhead of decompressing the format during startup.