XƎTEX, the Multilingual Lion: TEX meets Unicode and smart font technologies

Jonathan Kew
SIL International
Horsleys Green
High Wycombe HP14 3XL
England
jonathan_kew@sil.org

Abstract

Professor Donald Knuth's TEX is a typesetting system with a wide user community, and a range of supporting packages and enhancements available for many types of publishing work. However, it dates back to the 1980s and is tightly wedded to 8-bit character data and custom-encoded fonts, making it difficult to configure TEX for many complex-script languages.

This paper will introduce XƎTEX, a system that extends TEX with direct support for modern OpenType and AAT (Apple Advanced Typography) fonts and the Unicode character set. This makes it possible to typeset almost any script and language with the same power and flexibility as TEX has traditionally offered in the 8-bit, simple-script world of European languages. XƎTEX (currently available on Mac OS X, but possibly on other platforms in the future) integrates the TEX formatting engine with technologies from both the host operating system (Apple Type Services, CoreGraphics, QuickTime) and auxiliary libraries (ICU, TECkit), to provide a simple yet powerful system for multilingual and multiscript typesetting.

The most significant extensions XƎTEX provides are its native support for the Unicode character set, replacing the myriad of 8-bit encodings traditionally used in TEX with a single standard for both input text encoding and font access; and an extended `\font` command that provides direct access by name to all the fonts installed in the user's computer. It also provides a mechanism to access many of the advanced layout features of modern fonts.

Additional features that will also be discussed include built-in support for a wide variety of graphic file formats, and an extended line-breaking mechanism that supports Asian languages such as Chinese or Thai that are written without word spaces.

Finally, we look briefly at some user-contributed packages that help integrate the features of XƎTEX with the established LATEX system. Will Robertson's `fontspec.sty` provides a simple, consistent user interface in LATEX for loading both AAT and OpenType fonts, and accessing virtually all of the advanced features these fonts offer; Ross Moore's `xunicode.sty` is a package that allows legacy LATEX documents to be typeset using native Mac OS X fonts without converting the input text entirely to Unicode, by supporting traditional TEX input conventions for accents and other "special" (non-ASCII) characters.

— — * — —

Editor's note: This article is typeset in Adobe Garamond, with Andale Mono for the code examples, and processed on the author's Mac OS X machine with XƎTEX, as Unicode support was needed in several places.

What is XƎTEX?

XƎTEX[1] is an extension of the TEX processor, designed to integrate TEX's "typesetting language" and document formatting capabilities with the Unicode/ISO 10646 universal character encoding for all the world's scripts, and with the font technologies available on today's computer systems, including fonts that support complex non-Latin writing systems.

XƎTEX is in fact based on ε-TEX, and therefore includes a number of well-established extensions to TEX. These include additional registers (`\count`, `\dimen`, `\box`, etc.) beyond the 256 of each that TEX provides; various new conditional commands, tracing features, etc.; and of particular significance for multilingual work, the TEX–XƎT extension for bidirectional layout.

The TEX extensions inherited from ε-TEX are not discussed further here, as they are already described in the ε-TEX documentation[2], except to note that for right-to-left scripts in XƎTEX, it is necessary to set `\TeXXeTstate=1` and make proper use of the direction-changing commands `\beginR`, `\endR`, etc. Without these, there will still be some right-to-left behavior due to the inherent directionality defined by the Unicode standard for characters belonging to Hebrew, Arabic and similar scripts, but overall layout will not be correct.

XƎTEX was created in order to typeset materials — literacy and educational books, translated Scriptures, linguistic studies, dictionaries, etc. — in a wide range of languages and scripts, including lesser-known ones that are not adequately supported in most existing products. It inherits ideas, and even some code, from an earlier system called TEXGX that integrated TEX with the QuickDraw GX graphics system on older Macintosh operating systems.

[1] The name XƎTEX was inspired by the idea of a Mac OS X extension (hence the 'X' prefix) to ε-TEX; and as one of its intended uses is for bidirectional scripts such as Hebrew and Arabic, the name was designed to be reversible. The second letter should ideally be U+018E LATIN CAPITAL LETTER REVERSED E, but as few current fonts support this character, it is normal to use a reflected 'E' glyph. The name is pronounced as if it were written *zee-TEX*.

[2] E.g., *The ε-TEX Short Reference Manual*, `http://www.staff.uni-mainz.de/knappen/etex_ref.html`.

```
\font\Hoefler = "Hoefler Text" at 10pt
\Hoefler This is the Hoefler Text font.
```

This is the Hoefler Text font.

```
\font\VerdanaItal = "Verdana Italic" at 9pt
\VerdanaItal And this is Verdana Italic.
```

And this is Verdana Italic.

Figure 1: Accessing fonts installed natively on the host platform

Host platform fonts

For many users, one of the most significant features of XeTeX is that it makes use of the fonts installed in the operating system, just like mainstream GUI word processing and page layout programs. On Mac OS X, fonts in a number of major formats — in particular, TrueType (`.ttf`) fonts, and both TrueType- and CFF-flavored[3] OpenType (`.otf`) fonts, as well as legacy Macintosh resource file formats — can be installed in any of several **Library/Fonts** folders (system-wide, or per-user), and users expect these fonts to be available in all applications.

With a traditional TeX system, this is not the case. Because of its portable, platform-independent heritage, TeX knows nothing about the fonts installed in a particular operating system, or even about today's major font formats; it relies instead on `.tfm` files (an alien concept to the typical modern font user) to provide the metrics information needed for typesetting, and on output drivers that locate and use the actual font files containing glyph images. All these are specifically installed for TeX and associated tools, quite separately from font installation for the operating system or other applications. Many users find this a challenge, and do not feel confident to use fonts other than those provided with their TeX distribution. So there is a perception that TeX supports a very limited range of fonts. XeTeX aims to change this.

Font access in XeTeX Within a XeTeX document, it is trivial for users to typeset using whatever fonts they have on their computer system. If a Mac OS X user buys or downloads a `.ttf` or `.otf` font and installs it in the standard way with FontBook or by placing the file in **~/Library/Fonts**, the font can be used by just specifying it by name with a `\font` command, as in figure 1. No conversions, no auxiliary files, no TeX-specific installation or configuration; just tell XeTeX to use the font, and there it is. (Note that figure 1, like most examples in this paper, uses simple "plain TeX"-level commands; in the context of packages such as LaTeX or ConTeXt there would be higher-level commands designed to interact properly with the overall package.)

[3] CFF: Compact Font Format, the table type that holds PostScript glyph data in an OpenType font container.

When XeTeX is using "native" fonts from the operating system, it handles text in a slightly different way than standard TeX. Rather than treating each character individually, looking up its metrics (in a `.tfm` file), it collects "runs" (typically, but not always, complete words) and passes them to the font rendering subsystem as complete chunks of text. This is necessary in order to allow the font to implement features such as ligatures, cursive connections, contextual character substitutions or reordering, etc., which may be defined in AAT or OpenType fonts (see below). Such features may represent optional typographic refinements in Latin-based scripts, but in many Asian scripts they are essential for correct rendering.

Output device support Selecting fonts by name within the source document, and having the typesetting process find and use them when building paragraphs, is only half the story. Drivers that render TeX output onto a particular device also need to locate fonts — and in the traditional TeX world, the two stages rely on separate files, with typesetting requiring only `.tfm` files, and output requiring "real" fonts of some kind, e.g., `.pk` or `.pfb` files.

The current implementation of XeTeX creates output in an "extended DVI" format (`.xdv`), and this is then converted to PDF by a second process, `xdv2pdf`.[4] To generate PDF, `xdv2pdf` relies on the user's installed fonts in exactly the same way as the typesetting process. There is no separation between fonts as used during typesetting and those used for output.

Because the output format is effectively PDF (as the `.xdv` ⟶ `.pdf` conversion is automatically executed), XeTeX output can then be viewed or printed on any system or device where PDF is supported, using standard viewers and printer drivers.

Support for legacy TeX fonts In addition to using fonts installed natively in the operating system, XeTeX continues to support the use of existing fonts in the `texmf` tree, using `.tfm` files (for metrics) and `.pfb` fonts (Type 1 outlines, for rendering). When using such `.tfm`-based fonts, the results should be identical to those produced by standard TeX.

Note that the current `xdv2pdf` driver supports such legacy fonts only in `.pfb` format; there is no support, in particular, for `.pk` or other METAFONT-derived bitmap formats. There is also no `.vf` support at present.

The use of `.tfm`-based fonts is important partly for compatibility with existing documents that use these fonts, where a user might wish to take advantage of some XeTeX features without changing the overall look of the document. Perhaps more important, `.tfm`-based fonts are *required* for math mode, as TeX's math formatting makes use of detailed

[4] The default behavior is for the `xetex` process to automatically pipe its `.xdv` output through `xdv2pdf`, so that the default output format appears to be PDF.

metric information that comes from the METAFONT fonts and cannot readily be generated for the system's native fonts. This means that math typesetting continues to work unchanged in XeTeX; however, it also means that *for math*, the range of fonts available remains very limited.

Unicode support

TeX was originally designed for English typesetting, with characters needed for other (primarily European) languages supported via the `\accent` command and additional characters (such as ß and æ) provided in the Computer Modern fonts and accessed via control sequences, to escape the limitations of the ASCII character set. Many other languages and scripts have also been handled, using a variety of techniques including custom codepages and fonts, macros and "active" characters, and even preprocessors that implement specific complex scripts such as Devanagari.

The variety of TeX programming tricks available, together with the use of non-standard input and font encodings and similar techniques, have allowed many scripts to be typeset; however, they have also meant that the input text used for typesetting is often encoded in a non-standard way, unique to the particular TeX package in use, making for problems of data interchange with other systems. And the use of preprocessors and/or TeX macros to implement script behavior can easily conflict with other levels of macro programming (document markup and formatting control), making for complex and fragile systems.

The Unicode standard offers the possibility of a much simpler, cleaner multilingual system. In Unicode, every character needed for any script has (in principle) its own code, so there is no longer any need for multiple codepages, where the meaning of a particular character code depends on the input encoding or font in use. Nor is there any need for escape sequences or preprocessors to access characters that cannot be entered directly in the input; text in any language can be represented as simple character data. So XeTeX aims to extend TeX such that the standard character encoding used throughout the typesetting process, from text input to accessing glyphs in fonts, is Unicode.

Character codes The first step towards Unicode support in TeX is to expand the character set beyond the original 256-character limit. At the lowest level, this means changing internal data structures throughout, wherever characters were stored as 8-bit values. As Unicode scalar values may be up to U+10FFFF, an obvious modification would be to make "characters" 32 bits wide, and treat Unicode characters as the basic units of text.

However, in XeTeX a pragmatic decision was made to work internally with UTF-16 as the encoding form, making "characters" in the engine 16 bits wide, and handling supplementary-plane characters using UTF-16 surrogate pairs. This choice was made for a number of reasons:

- The operating-system APIs that XeTeX uses in working with Unicode text require UTF-16, so working with this encoding form avoids the need for conversion.
- A number of internal arrays in TeX are indexed by character codes. Enlarging these from 256 elements each to 65,536 elements seems reasonable; enlarging them to a million-plus elements each would dramatically increase the memory footprint of the system. To avoid this, a sparse array implementation might be used, but this would be significantly more complex to develop and test, and might well have a negative impact on typesetting performance.
- It seems unlikely, in any case, that there will be much need to customize these properties (see next section) for characters beyond Plane 0.

In view of these factors, XeTeX works with UTF-16 code units. Unicode characters beyond U+FFFF can still be included in documents, however, and will render correctly (given appropriate fonts) as the UTF-16 surrogate pairs will be passed to the font system.

Another possible route would have been to use UTF-8 as the internal encoding form, retaining the existing 8-bit code units used in TeX as characters. However, this would have made it impossible (without major revisions) to provide properties such as character category (letter, other printing character, escape, grouping delimiter, comment character, etc.), case mappings, and so on to any characters beyond the basic ASCII set; and it would also require conversion when Unicode text is to be passed to system APIs. Overall, therefore, UTF-16 was felt to be the most practical choice, and the appropriate TeX data structures were systematically widened.

Extended TeX code tables Along with widening character codes from 8 to 16 bits, the TeX code tables that provide per-character properties were enlarged to cover the range 0…65,535. This means that XeTeX has `\catcode`, `\sfcode`, `\mathcode`, `\delcode`, `\lccode` and `\uccode` values for each of the characters in Unicode's Basic Multilingual Plane. The default format files provided with XeTeX initialize the `\lccode` and `\uccode` arrays based on case mapping properties from the Unicode Character Database, so that the `\uppercase` and `\lowercase` primitives will behave as expected. Figure 2 shows how these extended code tables might be used.

Because these arrays are indexed by the individual code units of the UTF-16 data used in XeTeX, it is not possible to set these properties for characters beyond Plane 0. However, as these are mainly either CJK ideographs or characters of relatively obscure archaic scripts, it seems unlikely that there will be much need to change their `\catcode` values or apply case-changing commands.[5]

[5] Full use of math characters from Plane 1 is a separate issue, as math mode requires additional font and character properties.

```
\lowercase{DŽIN}
   džin
\uppercase{Esi eyama klɔ míaƒe nuvɔwo ɖa vɔ la}
   ESI EYAMA KLƆ MÍAƑE NUVƆWO ƉA VƆ LA
\catcode`王=\active \def王{...}
   % defines an individual Chinese character as a macro
```

Figure 2: Per-character code tables in XeTeX support Unicode

```
\patterns{
% break before or after any full vowel
1अ1
1आ1
1इ1
1ई1
1उ1
1ऊ1
1ऋ1
1ॠ1
% ...etc...
% break after vowel matra, but never before
2ा1
2ि1
2ी1
2ु1
2ू1
2ृ1
2ॄ1
2ॢ1
2ॣ1
% ...etc...
}
```

Figure 3: Hyphenation patterns using Devanagari letters

Input encodings While XeTeX is designed to work with Unicode throughout the typesetting process, users may well wish to typeset text that is in a different encoding. By default, XeTeX interprets input text as being UTF-8, converting multi-byte sequences to Unicode character codes appropriately, unless inspection of the file suggests that the text is UTF-16 (identified by a Byte Order Mark code, or by null high-order bytes in the initial 16-bit code units). Either way, the input is assumed to be valid Unicode.

Existing TeX documents that use purely 7-bit ASCII are of course also valid Unicode (UTF-8); but documents in 8-bit encodings such as Windows Latin or Cyrillic codepages, legacy Mac OS character sets, or East Asian double-byte encodings cannot be interpreted this way. They will typically contain byte sequences that are not legal in UTF-8; but even if the bytes are not ill-formed when read as UTF-8, they will not result in the intended characters.

To address this problem, XeTeX provides two commands that allow the input to be converted from a different encoding into Unicode:

`\XeTeXinputencoding "`***codepage-name***`"`
: changes the codepage for the current input file, beginning with the next line of text

`\XeTeXdefaultencoding "`***codepage-name***`"`
: sets the initial codepage for subsequently-opened input files (does not affect files already open for reading)

These commands allow input text in a non-Unicode encoding to be converted (using the converters from the ICU library[6]) into Unicode as it is read. Thus, text in Latin-1 or Big5 or Shift-JIS or many other encodings can be typeset directly using Unicode-compliant fonts.

Note that output text, whether in the transcript file or files written using `\openout` and `\write`, will always be UTF-8 Unicode, regardless of the codepage or encoding form of the input text.

Hyphenation support Along with other character-code-oriented parts of TeX, the hyphenation tables in XeTeX have been extended to support 16-bit Unicode characters. This means that it is possible to write hyphenation patterns that use any (Plane 0) Unicode letters, including non-Latin scripts as well as extended Latin (accented characters, etc.) Figure 3 shows a fragment from a Sanskrit hyphenation file created by a XeTeX user. With the traditional TeX approach to such scripts, using complex macros and preprocessing, it would be much more difficult to support hyphenation patterns.

The implementation of native Unicode font support in XeTeX, treating each word as a "black box" measured as a unit by the font subsystem, made it easy to form paragraphs of such "boxes" without extensive changes to the overall algorithms. However, TeX's automatic hyphenation mechanism, which comes into effect if it is unable to find satisfactory line-break positions for a paragraph on the initial attempt, applies to lists of character nodes representing runs of text within a paragraph to be broken into lines. But when using Unicode fonts in XeTeX, the line-break process sees "word nodes" as indivisible, rigid chunks.

Explicit discretionary hyphens may of course be included in TeX input, and these continue to work in XeTeX, as they become discretionary break nodes in the list of items making up the paragraph. The word fragments on either side, then, would become separate nodes in the list, and a line-break can occur at the discretionary node between them.

[6] `http://www.ibm.com/software/globalization/icu/`

In order to provide automatic hyphenation support, however, it was necessary to extend the hyphenation routine so as to be able to extract the text from a word node, use TEX's pattern-based algorithm (and exception list) to find possible hyphenation positions within the word, and then replace the original word node with a sequence of nodes representing the (possibly) hyphenated fragments, with discretionary nodes in between.

One more refinement proved necessary here: once the line-breaks have been chosen, and the lines of text are being "packaged" for final justification to the desired width, any unused hyphenation points are removed and the adjacent word (fragment) nodes re-merged. This is required in order to allow rendering behavior such as character reordering and ligatures, implemented at the smart-font level, to occur across unused hyphenation points. With an early release of XƎTEX, a user reported that OpenType ligatures in certain words such as *different* would intermittently fail (appearing as *different*, without the *ff* ligature). This was occurring when automatic hyphenation came into effect and a discretionary break was inserted, breaking the word node into sub-words that were being rendered separately.

Typographic features

Beyond simply allowing the use of any font on the user's system, XƎTEX also provides access to various advanced typographic features of AAT and OpenType fonts, so that users can take advantage of the full richness of these fonts.

AAT font features AAT (Apple Advanced Typography) is the native Mac OS X technology for advanced fonts that provide typographic layout information (besides simple glyph metrics). An AAT font may contain tables that define layout features such as ligatures, alternate glyph forms, swashes, etc. These features may be specified by the font as being enabled by default, in which case XƎTEX will automatically use them; or they may be optional features that are only used when explicitly turned on.

The font designer provides names, stored in the font itself, for any features that are intended to be controlled by the user. While there is a registry of known features, designers are free to implement and name new behaviors in their fonts, so the possible set of features and settings is open-ended.

The extended `\font` command in XƎTEX allows AAT font feature settings to be specified as a list of *feature* = *setting* pairs appended to the name of the font. Feature settings that are enabled by default can also be turned off, by prefixing the setting name with '`!`'. Figure 4 illustrates a few optional features available in the Apple Chancery font.

Vertical text with AAT fonts An additional attribute that can be specified for AAT fonts in XƎTEX is `vertical`. This causes the text rendering system to use vertical text-layout techniques, although it does not in itself re-orient the overall layout. Typically, glyphs will be rotated 90° counter-clockwise, like this, and laid out according to their vertical rather than horizontal metrics.

If this capability is combined with macros that rotate the text block as a whole, which is readily achieved through graphic transformations in the output driver (see figure 5), it becomes possible to typeset languages such as Chinese using a traditional vertical layout. Figures 6 and 7 show the same text formatted in horizontal and vertical styles. Note how certain glyphs such as the brackets do not undergo the same rotation as the rest of the text; the `vertical` attribute automatically gives the correct behavior here.

```
\font\x="Apple Chancery" at 10pt
\x The quick brown fox jumps over the lazy dog.
```

The quick brown fox jumps over the lazy dog.

```
\font\x="Apple Chancery:
  Design Complexity=Simple Design Level;
  Letter Case=Small Caps" at 10pt
\x The quick brown fox jumps over the lazy dog.
```

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.

```
\font\x="Apple Chancery:
  Design Complexity=Flourishes Set A" at 10pt
\x The quick brown fox jumps over the lazy dog.
```

The quick brown fox jumps over the lazy dog.

Figure 4: Selecting optional AAT font features

OpenType: optional features Like AAT fonts, OpenType fonts may also include layout features that can be enabled or disabled to affect the rendering of the text. Unlike AAT, there are no feature names provided in the font, only four-character "tags" (which are generally somewhat mnemonic). The expectation in OpenType is that all features will be officially registered with Microsoft and Adobe, and applications can then provide whatever user interface and names are needed for the features they choose to support.

XƎTEX takes a low-level approach, allowing feature tags to be used directly in the `\font` command in a similar way to AAT names; individual features can be turned on or off for any given font definition, using + or - with the four-character tag. Therefore, any features defined in the font can be used, even if not defined in the OpenType feature registry. (Macro packages could be used to provide more meaningful names; for example, the *fontspec* package for LATEX provides a unified interface for many registered features across both AAT and OpenType fonts.) Figure 8 shows a few examples of the use of OpenType feature tags to select alternate renderings of a font.

```
\newif\ifVertical \Verticaltrue % \Verticalfalse gives horizontal layout
\vsize=7in \hsize=4.5in \def\Vert{} % set up page size
\ifVertical % set parameters for vertical layout
  \hsize=7in \vsize=4.5in \def\Vert{:vertical} % attribute used in font defs
  % macro to rotate a box of Chinese text set with the "vertical" font attribute
  \def\ChineseBox#1{\setbox0=\vbox{\boxmaxdepth=0pt #1}\dimen0=\wd0 \dimen2=\ht0
    \vbox to \dimen0{\hbox to \dimen2{\hfil\special{x:gsave}\special{x:rotate -90}\rlap
        {\vbox to 0pt{\box0\vss}}\special{x:grestore}}\vss}}
  \def\ChineseOutput{\shipout \vbox{\ChineseBox{\makeheadline \pagebody \makefootline }}
      \advancepageno \ifnum \outputpenalty >-20000 \else \dosupereject \fi}
  \output={\ChineseOutput} \fi
\font\body="STKaiti\Vert" at 12pt \body
\font\bold="STHeiti\Vert" at 12pt  \font\title="STHeiti\Vert" at 18pt
\centerline{\title 三  国  演  义}
\bigskip \centerline{\bold〔明〕罗贯中}
% ...etc...
```

Figure 5: Using an AAT font attribute and graphic transformations to implement vertical typesetting

三　国　演　义

〔明〕罗贯中

词曰：

滚滚长江东逝水，浪花淘尽英雄。是非成败转头空：青山依旧在，几度夕阳红。

白发渔樵江渚上，惯看秋月春风。一壶浊酒喜相逢：古今多少事，都付笑谈中。

第一回

宴桃园豪杰三结义　斩黄巾英雄首立功

话说天下大势，分久必合，合久必分：周末七国分争，并入于秦；及秦灭之后， 楚、汉分争，又并入于汉；汉朝自高祖斩白蛇而起义，一统天下，后来光武中兴，传 至献帝，遂分为三国。推其致乱之由，殆始于桓、灵二帝。桓帝禁锢善类，崇信宦官 。及桓帝崩，灵帝即位，大将军窦武、太傅陈蕃，共相辅佐；时有宦官曹节等弄权， 窦武、陈蕃谋诛之，机事不密，反为所害，中涓自此愈横。

Figure 6: Chinese text in horizontal format

三　国　演　义

〔明〕罗贯中

词曰：

滚滚长江东逝水，浪花淘尽英雄。是非成败转头空：青山依旧在，几度夕阳红。

白发渔樵江渚上，惯看秋月春风。一壶浊酒喜相逢：古今多少事，都付笑谈中。

第一回

宴桃园豪杰三结义　斩黄巾英雄首立功

话说天下大势，分久必合，合久必分：周末七国分争，并入于秦；及秦灭之后， 楚、汉分争，又并入于汉；汉朝自高祖斩白蛇而起义，一统天下，后来光武中兴，传 至献帝，遂分为三国。推其致乱之由，殆始于桓、灵二帝。桓帝禁锢善类，崇信宦官 。及桓帝崩，灵帝即位，大将军窦武、太傅陈蕃，共相辅佐；时有宦官曹节等弄权， 窦武、陈蕃谋诛之，机事不密，反为所害，中涓自此愈横。

Figure 7: Chinese text in vertical format

OpenType: optical sizing Some OpenType font families include multiple faces designed for use at different sizes; for example, the Adobe Brioso Pro family includes Caption, Text, Subhead, Display, and Poster faces, each optimized for a different range of point sizes. If the full collection of fonts has been installed, XeTeX will use the OpenType "size" feature to automatically select the appropriate face for the point size used, as shown in figure 9. Generally, this automatic behavior is helpful; however, it can be overridden if necessary by using a `/S=`*optical-size* modifier on the font name. Figure 10 shows several different optical sizes of Brioso used at the same physical size, making the design difference between the faces more apparent to the eye.

OpenType: script and language In addition to optional typographic features, OpenType fonts may include layout features that are necessary for the correct rendering of complex writing systems such as Arabic or Indic scripts. To apply these features, it is necessary to have a "shaping engine" that applies the appropriate feature tags to individual characters of the text. There are specific rules for each supported script, and complex scripts will only render properly

```
\font\x="Brioso Pro" \x Hello World! 0123456789
```
Hello World! 0123456789
```
\font\x="Brioso Pro:+smcp" \x Hello World! 0...
```
HELLO WORLD! 0123456789
```
\font\x="Brioso Pro:+sups" \x Hello World! 0...
```
H^ello W^orld! ^0123456789
```
\font\x="Brioso Pro Italic:+onum" \x Hello W...
```
Hello World! 0123456789
```
\font\x="Brioso Pro Italic:+swsh,+zero" \x H...
```
Hello World! Ø123456789

Figure 8: Selecting OpenType feature tags

```
\font\x="Brioso Pro" at 7pt \x Hello World!
```
Hello World! (Brioso Pro Caption)
```
\font\x="Brioso Pro" at 10pt \x Hello World!
```
Hello World! (Brioso Pro Text)
```
\font\x="Brioso Pro" at 16pt \x Hello World!
```
Hello World! (Brioso Pro Subhead)

Figure 9: Automatic optical sizing

if the correct engine is specified in the `\font` command, as illustrated in figure 11. (Note that this is different from the situation with AAT fonts, where complex rendering behavior is programmed entirely in the font tables, and no script-specific engine is needed.)

OpenType fonts may also support multiple “language systems” to handle differences in the appropriate rendering for different languages. For example, many serifed Latin fonts include an *fi* ligature, and this will normally be enabled by default. However, Turkish makes a distinction between *i* and *ı* (dotless *i*). Using an *fi* ligature typically causes this distinction to be lost, and therefore this ligature must be disabled in the Turkish language system. Another example of language-specific behavior occurs in Vietnamese, where the positioning of multiple diacritics on a base character differs from the default vertically-stacked behavior that would be expected elsewhere. When loading an OpenType font in XƎTEX, the desired language tag can be included in the `\font` command to control the behavior, as shown in figure 12.

```
\font\x="Brioso Pro/S=7" at 12pt\x Hello World!
```
Hello World! (Brioso Pro Caption)
```
\font\x="Brioso Pro/S=10" at 12pt\x Hello World!
```
Hello World! (Brioso Pro Text)
```
\font\x="Brioso Pro/S=16" at 12pt\x Hello World!
```
Hello World! (Brioso Pro Subhead)

Figure 10: Overriding normal optical sizing

```
\font\x="Code2000" \x العربي हिन्दी
```
العربي हिन्दी *default (Latin) features only; incorrect rendering of both scripts*
```
\font\x="Code2000:script=arab" \x العربي
```
العربي *correct Arabic script rendering*
```
\font\x="Code2000:script=deva" \x हिन्दी
```
हिन्दी *correct Devanagari script rendering*

Figure 11: Specifying OpenType shaping engines using the `script=...` feature

```
\font\Brioso="Brioso Pro"
\Brioso …gelen firmaları…tarafından…
```
...gelen firmaları...tarafından...
```
\font\BriosoTrk="Brioso Pro:language=TRK"
\BriosoTrk …gelen firmaları…tarafından…
```
...gelen firmaları...tarafından...
Turkish requires the i/ı distinction maintained

```
\font\D="Doulos SIL/ICU"
\D cung cấp một con số duy nhất cho mỗi ký tự
```
cung cấp một con số duy nhất cho mỗi ký tự
```
\font\V="Doulos SIL/ICU:language=VIT"
\V cung cấp một con số duy nhất cho mỗi ký tự
```
cung cấp một con số duy nhất cho mỗi ký tự
Vietnamese uses different diacritic positioning

Figure 12: Using alternate language systems in OpenType fonts to achieve correct rendering

Font mappings In addition to the font-specific AAT and OpenType features that can be included in a `\font` command, XƎTEX has a general-purpose mechanism known as “font mappings” that can be applied to any native font.

To understand the purpose of font mappings, consider TEX input conventions such as `---`, which normally generates an em-dash, or ` `` `, which generates an opening double quote. These conventions are not built into TEX, nor are they generally implemented in TEX macros (like most other “extended” characters); rather, they are implemented as ligatures in the Computer Modern fonts, and similar ligature rules have been created in most other fonts configured for use with TEX.

However, these ligatures, unlike standard typographic ligatures such as *fi*, are not generally known or used

```
LHSName "TeX-text"
RHSName "UNICODE"

pass(Unicode)
U+002D U+002D <> U+2013 ; -- -> en dash
U+002D U+002D U+002D <> U+2014 ; --- -> em dash

U+0027 <> U+2019 ; ' -> right single quote
U+0027 U+0027 <> U+201D ; '' -> right dbl quote
U+0022  > U+201D ; " -> right double quote

U+0060 <> U+2018 ; ` -> left single quote
U+0060 U+0060 <> U+201C ; `` -> left dbl quote

U+0021 U+0060 <> U+00A1 ; !` -> inv. exclam
U+003F U+0060 <> U+00BF ; ?` -> inv. question
```

Figure 13: The `tex-text` font mapping

```
\def\SampleText{Unicode - это уникальный
 код для любого символа, независимо от платформы,
 независимо от программы, независимо от языка.}
\font\gen="Gentium"
\gen\SampleText\par
```

Unicode - это уникальный код для любого символа, независимо от платформы, независимо от программы, независимо от языка.

```
\font\gentrans="Gentium:mapping=cyr-lat-iso9"
\gentrans\SampleText\par
```

Unicode - èto unikal'nyj kod dlâ lûbogo simvola, nezavisimo ot platformy, nezavisimo ot programmy, nezavisimo ot âzyka.

Figure 14: Using a font mapping to render the same text in its native script and transliterated

outside the TeX world. They were designed as a convenient workaround for limitations of the character set that could be entered on typical keyboards. But we cannot expect general-purpose fonts from outside the TeX world to implement these ligatures. Therefore, if a XeTeX user working with a standard Unicode font enters ``` ``Help---I'm stuck!'' ```, the result is likely to be something like ``*Help---I'm stuck!''*, which is not what was intended.

One solution is to convert the input text to directly use the desired Unicode characters for quote marks, dashes, etc., but this may not be convenient where there are large amounts of pre-existing text. Even for new text, experienced TeX typists may be more comfortable continuing to use these conventions rather than learning new key sequences, or document portability between XeTeX and standard TeX may require that they be used.

XeTeX's font mappings can solve this issue. A font mapping is a transformation, expressed as mapping rules that convert Unicode characters or sequences from an "input" form (that found in the document text) to an "output" (the character or characters to be rendered from the font). Such mappings are written in the TECkit mapping language.[7] A `\font` command may include a `mapping=`*filename* qualifier, and XeTeX will then apply the given mapping as part of the text rendering process when using that font. An example `tex-text` mapping is included with XeTeX to implement the common ligatures found in Computer Modern fonts; figure 13 shows the TECkit source of this mapping file. If this mapping is loaded along with a standard Unicode font, then the TeX-style input text ``` ``Help---I'm stuck!'' ``` will render as expected: *"Help—I'm stuck!"*.

[7] `http://scripts.sil.org/teckit/`

While font mappings were originally implemented to provide compatibility with TeX typing conventions, they can be used in other ways, too; figure 14 shows an example where the same input text is printed both in its original form and in Latin transliteration, using a Cyrillic/Latin transliteration mapping associated with the font.

Asian-language linebreaking

A number of east and south-east Asian languages, such as Chinese, Japanese, Thai, and others, are normally written without word spaces. The only spaces in the text may be between phrases or sentences, or even entire paragraphs may be lacking any space characters. Hyphenation is also not used in many of these languages. This presents a problem for line-breaking, as TeX normally expects to find interword glue where line-breaks can be attempted.

Line-break positions To support typesetting text in such languages, XeTeX includes a feature known as "locale-based line-breaking", based on the Unicode line-break algorithm implemented in the ICU library. The command `\XeTeXlinebreaklocale="`*locale-code*`"`, where the *locale-code* is a standard locale (language/region) code, tells XeTeX to look for possible line-break positions according to the rules of the given locale; the paragraph can then be broken at these places despite the lack of spaces or hyphenation rules.

Justification In addition to the problem of finding legitimate line-break positions, the lack of inter-word glue also makes it difficult for TeX to justify the lines. One option, of course, is ragged-right typesetting, and this may be the appropriate solution if a rigid character grid (as sometimes seen in Chinese, for example) is to be maintained. However, another option is to set the parameter `\XeTeXlinebreakskip` to a slightly stretchable glue value.

```
\def\thaitext{%
โดยพื้นฐานแล้ว, คอมพิวเตอร์จะเกี่ยวข้องกับเรื่องของตัวเลข.
คอมพิวเตอร์จัดเก็บตัวอักษรและอักขระอื่นๆ
โดยการกำหนดหมายเลขให้สำหรับแต่ละตัว.
ก่อนหน้าที่ Unicode จะถูกสร้างขึ้น, ได้มีระบบ encoding
อยู่หลายร้อยระบบสำหรับการกำหนดหมายเลขเหล่านี้.}
\font\thai="Thonburi" at 10pt
\thai \thaitext
```

โดยพื้นฐานแล้ว, คอมพิวเตอร์จะเกี่ยวข้องกับเรื่องของตัวเลข. คอมพิวเตอร์จัดเก็บตัวอักษรและอักขระอื่นๆ โดยการกำหนดหมายเลขให้สำหรับแต่ละตัว. ก่อนหน้าที่ Unicode จะถูกสร้างขึ้น, ได้มีระบบ encoding อยู่หลายร้อยระบบสำหรับการกำหนดหมายเลขเหล่านี้.

Thai text with spaces only between phrases

```
\XeTeXlinebreaklocale "th"
\XeTeXlinebreakskip=0pt plus 1pt
\thai \thaitext
```

โดยพื้นฐานแล้ว, คอมพิวเตอร์จะเกี่ยวข้องกับเรื่องของตัวเลข. คอมพิวเตอร์จัด เก็บ ตัว อักษรและ อักขระ อื่นๆ โดย การ กำหนด หมาย เลข ให้ สำหรับ แต่ ละ ตัว. ก่อนหน้าที่ Unicode จะ ถูก สร้าง ขึ้น, ได้ มี ระบบ encoding อยู่ หลาย ร้อย ระบบ สำหรับ การกำหนดหมายเลขเหล่านี้.

Using locale-based line-breaking to improve results

Figure 15: Line-breaking and justification without word spaces

XƎTEX will then insert this glue at each *potential* break position found by the line-break algorithm, which makes the overall text slightly stretchable and allows fully justified setting. Figure 15 illustrates the use of the Asian line-breaking parameters.

There is also a parameter `\XeTeXlinebreakpenalty` that can be set to control the desirability of inter-character breaks found by the Unicode algorithm, as compared to normal breaks at other penalties and glue.

Built-in graphics support

TEX traditionally knows nothing about graphics, leaving this to output drivers and merely passing along information from `\special` commands. It is left to macro packages and users to determine the amount of space that an included image occupies; the `\special` that causes the image to be included by the driver does not itself take any space during the typesetting process.

XƎTEX provides an alternative approach, adding primitive commands that actually load graphic files during typesetting. The advantage of this is that the typesetting process can know the size of the image; typically, it is loaded into an `\hbox`, and macros can then examine the `\wd` and `\ht` of that box to make decisions about layout, or to re-load the image with different scaling, etc.

```
\centerline{%
 \hbox{\XeTeXpicfile "unicode-book.jpg"
   scaled 100}\quad
 \hbox{\XeTeXpicfile "unicode-book.jpg"
   scaled 100 xscaled 2000}\quad
 \hbox{\XeTeXpicfile "unicode-book.jpg"
   scaled 100 rotated 90}}
```

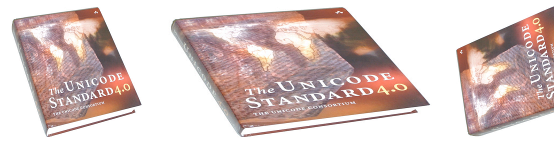

Figure 16: Including graphics in a XƎTEX document

QuickTime-based graphics The XƎTEX primitive command `\XeTeXpicfile "`*filename*`"` locates and includes the named graphic file, which may be in any format recognized by the QuickTime library on Mac OS X. This includes common image formats such as `.jpg`, `.bmp`, `.tiff`, `.png`, and many others. A number of keywords such as `width`, `height`, `scaled`, and `rotated` may be used after the filename to transform the image. Figure 16 shows some simple examples of image inclusion.

PDF documents One of the formats supported by the `\XeTeXpicfile` command is `.pdf`; however, if a PDF graphic is included in this way, it will be rendered as a raster image at relatively low resolution. It is better to use an alternative command, `\XeTeXpdffile`, which includes the specified PDF in its native form, complete with vector graphics, embedded fonts, etc. `\XeTeXpdffile` also supports an additional keyword `page` to select the required page from a multi-page PDF document.

Note that there is a `xetex.def` driver available for the standard LATEX `graphics.sty` and `graphicx.sty` packages; this driver will automatically use the XƎTEX primitives to implement the higher-level `\includegraphics` command, and will choose the proper XƎTEX function depending on the type of graphic file.

LATEX packages

Many users like to combine the Unicode and font support of the XƎTEX engine with the document markup and formatting features of LATEX. In most cases, this works well; the exceptions typically involve LATEX packages dealing with input and font encodings (which are generally superfluous in a Unicode-based process) or packages that depend on the features of a particular output driver (such as drawing packages that rely on `dvips` or `dvipdfm` `\specials`, or on pdfTEX extensions). In some cases, such packages may need to be adapted to work with the `xdv2pdf` driver; in others, the output driver features needed may not currently be available.

```
\usepackage{fontspec} % load fontspec.sty
\setmonofont[Scale=0.8]{Andale Mono WT J}
  % use scaled Andale Mono for \tt
\defaultfontfeatures{Mapping=tex-text}
  % load the tex-text font mapping by default
\setromanfont{Adobe Garamond Pro}
  % use Garamond Pro as \rm, etc
```

Figure 17: Use of `fontspec.sty`, from the preamble of this document

In addition to the `xetex.def` driver files for the standard LaTeX graphics and color packages, allowing these to be used with the XeTeX engine, two important packages written specifically for XeTeX deserve mention.

fontspec The `fontspec.sty` package, written by Will Robertson, provides a high-level interface to native Unicode fonts in XeTeX, integrating them with the LaTeX font selection mechanism, and supporting a wide range of features in both AAT and OpenType fonts. Extensive documentation is available with the package; figure 17 shows a simple excerpt from the preamble of this document. These few lines are sufficient to set up all the typefaces needed for this document, except those used within figures to illustrate specific points. Note that there are no auxiliary `.tfm`, `.fd`, `.sty`, or other TeX-specific files associated with the fonts used here; they are simply installed in the **~/Library/Fonts** folder in the standard Mac OS X manner.

xunicode To improve support for standard LaTeX documents when using Unicode fonts, Ross Moore has provided `xunicode.sty`. This package reimplements many of the control sequences used in LaTeX for accents, symbols, and other "special" characters, mapping them to the correct Unicode codepoints instead of to their locations in traditional TeX fonts. This allows documents that use these symbols via their LaTeX names to run unchanged under XeTeX, with the correct Unicode characters being rendered in the output.

XeTeX and ConTeXt

While LaTeX is probably the macro package most commonly used with XeTeX, it is also possible to use ConTeXt. My understanding is that the standard ConTeXt distribution includes an option to use the XeTeX engine in place of the default pdfTeX. A brief example of how XeTeX font support can be used in ConTeXt is shown in figure 18. There is further information on the ConTeXt Wiki site,[8] from which this example was copied.

```
\definedfont["Hoefler Text:mapping=tex-text;
  Style Options=Engraved Text;
  Letter Case=All Capitals" at 24pt]
Big Title
```

BIG TITLE

Figure 18: Loading a native Unicode font in ConTeXt

Future directions

In conclusion, a few comments on the possible future of XeTeX. The system has been publicly available for about 18 months as of the time of writing, and has been used for a wide range of document types and languages. While it remains a "work in progress", it appears to work reliably for most users, within the limitations of its design.

Besides on-going bug fixes and minor features, there are several major enhancements that could be undertaken to further improve XeTeX:

- Enhanced PDF back-end, via one of several approaches:
  - leverage improved PDF support in Mac OS X 10.4
  - new `xdv2pdf` driver based on `dvipdfmx`
  - integration with pdfTeX output routine
- True Unicode math support:
  - requires extensions to `\mathchar` etc., and underlying structures
  - also requires extended (at least 16-bit) font metric format
  - may be possible to make use of code from Omega/Aleph
- XeTeX for non-Mac OS X platforms:
  - should include full integration with TeX Live sources

Assistance towards implementing any and all of these ideas, or others, is most welcome! The XeTeX source code is currently available in a Subversion repository at `svn://scripts.sil.org/xetex/TRUNK`; this URL may change at some point, but the XeTeX web pages at `http://scripts.sil.org/xetex` should always indicate where to look.

[8] See `http://wiki.contextgarden.net/XeTeX` and `http://wiki.contextgarden.net/Fonts_in_XeTeX`.